NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COMPUTER-AIDED RECOGNITION OF MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS

by

Luiz Alberto Lisboa da Silva Cardoso

December 1999

Thesis Advisor:	Neil C. Rowe
Second Readers:	Robert B. McGhee
	Roberto Cristi

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	**2. REPORT DATE** December 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
COMPUTER AIDED RECOGNITION OF MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS

5. FUNDING NUMBERS

6. AUTHOR(S)
Cardoso, Luiz Alberto L. S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Brazilian Naval Commission
5130 MacArthur Blvd. N. W.
Washington, D.C. 20016-3344

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense, the U.S. Government or the Brazilian Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Aerial image acquisition systems are producing more data than can be analyzed by human experts. Most of the images produced by remote sensing satellites, including military ones, never get seen or inspected. In this work, automated detection and recognition of buildings in aerial photos is explored. Connectivity analysis is performed on graphs derived from line segment representations of the original images, obtained with the use of the Radon Transform. The model is experimentally validated using 2-meter panchromatic aerial photographs from the National Aerial Photography Program (NAPP), which provide a marginally adequate resolution for the recognition of small buildings.

14. SUBJECT TERMS
Aerial photograph analysis, pattern recognition, imagery intelligence

15. NUMBER OF PAGES
161

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239.18

COMPUTER AIDED RECOGNITION OF MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS

Luiz Alberto Lisoba da Silva Cardoso
Lieutenant Commander, Brazilian Navy
B.S.E.E., Military Institute of Engineering, 1985
M.S.E.E., Catholic University of Rio de Janeiro, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: [illegible]
Luiz Alberto Lisoba da Silva Cardoso

Approved by: Neil C. Rowe
Neil C. Rowe, Thesis Advisor

Robert B. McGhee
Robert B. McGhee, Second Reader

[illegible]
Roberto Cristi, Second Reader

Dan C Boger
Dan Boger, Chairman
Department of Computer Science

ABSTRACT

Aerial image acquisition systems are producing more data than can be analyzed by human experts. Most of the images produced by remote sensing satellites, including military ones, never get seen or inspected. In this work, automated detection and recognition of buildings in aerial photos is explored. Connectivity analysis is performed on graphs derived from line segment representations of the original images, obtained with the use of the Radon Transform. The model is experimentally validated using 2-meter panchromatic aerial photographs from the National Aerial Photography Program (NAPP), which provide a marginally adequate resolution for the recognition of small buildings.

DISCLAIMER

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author would like to acknowledge the extensive and unconditional support received by his sponsor agency, the Brazilian Navy, who provided not only complete financial support along his graduate studies, but also and more importantly the concrete motivation to purse new frontiers of knowledge.

More than to any other person, the author also wants to thank Prof. Neil C. Rowe, his research supervisor, for his patience, dedication and thoroughness in reviewing the text and the technical content of this work. The same qualities, balanced with freedom of action and creativity, were present in all his outstanding classes.

From the not a little less bright classes taken from Professors Yutaka Kanayama, Robert McGhee, Roberto Cristi, Monique Fargues, Ralph Hippenstil, Thomas Wu, Craig Rassmussen, Harold Fredericksen and Carlos Borges came the remaining background specialization in Artificial Intelligence, Signal Processing, Algorithms and Mathematics necessary to accomplish the task (some would say that everything is Mathematics, anyway). Not diminishing the contribution of some other excellent professors and lecturers in other areas, the author expresses a very special recognition to those above.

Two other important acknowledgements go to the NPS staff. First to Mr. Gary Rediske and his team, for their silent and competent work that kept the main computational resources used in the development of this work in good shape and at virtually 100% availability, both hardware & software. The second goes to the Computer Science Department assistant, the efficient and caring friend Mrs. Jean Brennan, who always handled his administrative affairs timely and with no flaws.

Finally, the author declares his deep gratitude to his family and many friends from whom he learned perhaps more and more valuable things along the way than from either his graduate studies or this thesis itself.

I. INTRODUCTION

A. AERIAL PHOTOGRAPHY

Since the advent of photography last century, it has been used as a descriptive resource for a large variety of urban constructions and other human-made structures. Since the nineteenth century, aerial photographs – obtained from the top of nearby hills or balloons, from kites or airplanes, whatever the technology could offer as an elevated platform for a camera – were always highly regarded as a descriptive tool [Ref. 1].

More recently, artificial satellites provide the ultimate platform for a camera: Permanently in the skies, from a high latitude orbit, a satellite is able to periodically cover virtually any point of the globe every few days. From a military perspective, a camera hundreds of thousands kilometers above ground is much more convenient, safer and discreet than a manned flight within reach of enemy reaction. Also, orthorectified images can be easily produced from the raw pictures collected, since orbits, position in orbit and attitude are controlled and precisely known when a picture is captured. That means that it is possible to measure the geometry of the area photographed to a high degree of accuracy.

Modern remote sensing satellites are equipped with high-definition electronic cameras and high-bandwidth communication ports that enable ground control stations to receive images in digital format and instantaneously relay them to designated locations continuously. This opens new fronts for the analysis of aerial photographs: Beyond quality and availability, the latency in the capture process is now minimal. And by being in digital format, the information can be easily stored, retrieved and made available to a computer program.

This is appealing because analysis of an aerial photo by a human expert is slow, prone to error and often infeasible due to lack of sufficient manpower. The automation of photographic analysis is one of the main research topics in remote sensing today. Most of the results concern high-level categorization of terrain and the production of digital maps. The military uses digital terrain modeling and general electronic cartography for Command, Control, Communications, Computers and Intelligence (C^4I) systems.

Semantic contextual analysis of photographs remains still an area of open investigation, since the only effective tools for it today are well-trained human experts. The experts perform a set of actions: (i) Pixels of the image are grouped into entities; (ii) entities are recognized; (iii) relationships are established; and (iv) conclusions are drawn from the overall scenario. Semantic interpretation yields not data and statistical features but conclusions and facts.

B. MILITARY APPLICATIONS OF AERIAL PHOTOGRAPHS ANALYSIS

Typical military uses of aerial photography include:

Tactical area surveillance, monitoring over a geographic area for detecting and locating targets that are predictable in nature (for instance, ships, tanks, aircrafts, and personnel). Target activity can also be tracked along time.

Strategic wide area surveillance, monitoring over a large geographic area for interesting or unexpected or not restrictively defined events that might have long-term military relevance. Many events in this class either take time, like building construction and supply relocation, or have long lasting detectable consequences, like natural catastrophes. Such monitoring was important recently for NATO operations in Kosovo [Ref. 2].

Target analysis or tactical survey, analysis performed on a limited amount of information concerning some specific area or object and its surroundings, for force evaluation or mission planning.

Damage assessment survey, a special type of tactical survey performed after a strike or attack, estimating damage produced to targets. A previous tactical survey should be available for comparison. Accurate damage assessment is important since information and impressions collected during the battle may be misleading or false, since often the damage is less intense than is believed.

Special-forces mission-planning survey, analysis to support the deployment of special forces. These are missions where direct contact or exposure to the enemy is implied and the area surveyed is usually enemy-controlled and may not be easily accessed by means other than photographic. Activities in these missions may include guerilla warfare, evasion and escape, subversion, sabotage, and other operations of requiring low visibility or a covert nature. Photointerpretation requirements are demanding with respect to accuracy, level of detail, and delivery timing.

The analysis associated with these tasks is similar in nature. The challenge to the expert is the time between when images are acquired and when conclusions must be derived (especially in the last two tasks). Analysis can be complex and an intense discussion between experts and mission command often occurs, making it desirable that the experts be locally available.

C. VISION: THE NEED FOR AUTOMATED ANALYSIS

> **Well less than half of the pictures taken by our satellites ever get looked at by human eyes or by any sort of mechanized device or computerized device ...and there is no plan at the present to build up an image analytic capability** - John Mills, staff director for the U.S. House Intelligence Committee (declaration published on March 26, 1999). [Ref. 3].

The above quotation suggests a gap between the investment of billions of dollars by the United States on hardware for acquiring high-quality imagery and the necessary analytical ability. Was this a mistake? No, because easier problems should be solved first in a technical area. But the current situation urges for the development of automated analysis techniques and tools for military and intelligence needs because:

(i) The analytical manpower available today is unable to use all the costly large data sets generated by the latest generation of collecting platforms, and data collection rates continue to increase.

(ii) Response-time requirements for analysis are continually becoming shorter for military applications, and such applications must always be judged on a competitive basis.

(iii) Automated analytic systems would mostly compensate the absence of an expert on-site, giving a chance to interactively question a hypothesis. If experts are available on-site, automated tools could still enhance their productivity.

D. THE CONTRIBUTION OF THIS WORK

Analysis of aerial photography by a hierarchical approach much like that of experts requires the following actions:

(i) Detect man-made structures in aerial photos;

(ii) Recognize these structures;

(iii) Establish relationships between these structures;

(iv) Infer useful assertions based on the relationships found;

(v) Answer user questions regarding the image.

This work investigates the use of computer vision techniques for the task of aerial photography analysis, focussing on the study and validation of some concepts in (i) and (ii) above.

The investigation was concentrated on the recognition of buildings, among the most important features militarily. Most of the existing techniques require clear, high-definition images. We chose to work with lower-definition images, hoping that robust algorithms would later prove themselves useful for extracting more detailed information within entities.

The algorithms developed were tested and evaluated using 2-meter panchromatic images from commercial sources. This is about the lower limit of resolution for human experts to correctly identify buildings. Benchmarking under these conditions gives a more realistic comparison of the algorithms qualities to the human skills. Results obtained indicate promising techniques that may be applied in future automated analysis systems.

Chapters II and III provide the reader with essential background in aerial photography and computer vision methods. Our program for photography analysis and building recognition is detailed in Chapter IV. Results obtained from the experimentation are summarized in Chapter V. Chapter VI contains concluding remarks about the investigation conducted.

THIS PAGE INTENTIONALLY LEFT BLANK

II. AERIAL PHOTOGRAPHY

Though aerial observation has military value, its also has value to city planning, urban development monitoring, map building, legal disputes, and other civilian activities [Ref. 4]. Some companies in the business of aerial photography have existed for more than fifty years [Ref. 5]. Currently there is a large demand for aerial photographs, and the projected market for the next decade is billions of dollars.

Aerial photography can have many attributes. Pictures taken of the same location may substantially differ due to incidental reasons, like current illumination and weather. However some other more conspicuous structural reasons will exist as a consequence of the different processes and camera parameters being used. In a first simplifying approach, the quality of an image will depend on spectral information, camera attitude and resolution, and the elevating platform.

A. SPECTRAL INFORMATION

Images can be *multispectral*, if the energy of different spectral bands is registered separately, or *panchromatic*, if only one data value is registered per pixel. Commonly multispectral components are mapped to the basic color channels (red, green and blue). If, however, a single one-dimensional value is captured per pixel, representing the total energy along the spectrum, the data set can be visualized by the gray tones. The selection of band sensitivity is fundamental, for the phenomena being photographed may be better visible in certain wavelengths. The lower the wavelength, the higher the potential resolution. Seeing through atmospheric haze, however, is facilitated at infrared wavelengths due to the light scattering of air and water at visible-light wavelengths. But the highest-resolution images are obtained by merging the information of multiple spectral bands. This is also why pattern analysis based on apparently less informative gray tones images can still be worthwhile, what we study in this work.

B. CAMERA ATTITUDE

Aerial photographs can be taken from different angles with respect to the earth surface. Planimetric errors are introduced proportional to the cosine of the deviation angle from directly above. Images taken up to 15° from straight down will produce less than 3.5% relative error in measurements of lengths and less than 2° error in measurements of right angles, as shown in Table 1.

δ, Deviation angle from straight above	Relative length error due to parallax effect	Max angular error for right angles
5°	-0.38 % to 0	±0.22°
10°	-1.5% to 0	±0.88°
15°	-3.4% to 0	±2.0°
20°	-6.0% to 0	±3.6°

Table 1: Maximum distortions in aerial photos caused by deviation from straight above, not considering the curvature of the Earth.

C. ORTHORECTIFICATION

Angle preservation and local planimetric fidelity can be obtained in every point of the image if a transformation called *orthorectification* is applied to it. This distorts locations in an aerial photo based on camera and viewpoint parameters. Of the images used in this work, those from the National Aerial Photography Program are not orthorectified, while those from the SPIN-2 imagery are. But the NAPP images were not a problem because the deviation angle is kept less than 4 degrees, insignificant to the results.

D. ELEVATING PLATFORM

Nowadays, the platform is an airplane or artificial satellite. The same camera on board a lower altitude aircraft (500m to 20km typical altitude) will give much higher-resolution pictures than on a satellite (800 km typical orbit height). But satellites are often preferred, because they offer a worldwide, safe, and concealed coverage, invaluable in military operations. Their positioning and attitude control are not subject to wind and other mechanical disturbances that may affect aircraft.

E. MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS

Example of these are buildings, roads, harbors, and runaways. Buildings are usually the most relevant man-made structures in aerial photographs. Their detection is valuable because most of strategic human activity occurs in, or in association with, a building of some sort. Also, as they do not move, they serve as good references for the relative position of other type of objects. Experts detect their presence based on the straight edges and right angles of their contours, often accompanied by contrast to the background.

F. LIMITATIONS INDUCED BY RESOLUTION

1. Spatial Resolution

An object can be detected by an imaging system provided it radiates enough luminance in the direction of the camera. So resolution cannot be defined as the minimum size that an object must have to be detected. *Spatial resolution* of an image, expressed in meters, is defined as the minimum distance between two individually detectable point objects at which they still can be distinguished. The *sample definition* of an image, the distance in meters between pixels, should not be confused with the spatial resolution. To preserve information of the image, the sample definition employed in its digitization should be smaller than the resolution. However, increasing the sample definition much beyond that will not increase the intrinsic image definition.

Detecting an object will not guarantee its recognition. The characteristic size, luminance and contrast of buildings in aerial images makes them typically detectable at 10m resolution. For an expert to assert that something is a building (recognition) requires a resolution about two times better (5m). For the classification of buildings, the resolution should be better than 2.5m. [Ref. 6].

2. Radiometric Resolution

Radiometric resolution is the number of quantization levels for the luminance of each pixel. It is commonly expressed in bits. Although the dynamic range of human eye sensitivity is about 10^9 [Ref. 7], the maximum number of gray levels that can be perceived is around 30 to 60 (roughly 6 bits). Improving the radiometric resolution of a digital panchromatic image further than that will not affect its analysis by human experts. Nonetheless, it helps to have raw images digitized at higher resolutions so that the original levels can be mapped into a good 6-bit presentation range. Typical resolution of commercial imagery is 8 bits.

G. HIGH RESOLUTION COMMERCIAL IMAGERY USED IN THIS WORK

New commercial high-resolution imagery satellites such as the IKONOS, just launched, are predicted to be operational within the next few months, delivering 1-meter resolution panchromatic and 4-meter multi-spectral data [Ref. 8]. Meanwhile, high-resolution imagery comes from either airborne photography or formerly classified 2-meter satellite imagery from the seventies and eighties which are being released to public under commercial agreements.

The aerial photographs used in this work were panchromatic images from two sources: The National Aerial Photography Program (NAPP) of the U.S. Geological Survey (USGS) and the SPIN-2 from Sovinforsputnik consortium.

The NAPP aerial photographs are taken on roughly a 6-year cycle, covering the entire continental U.S. They are shot with a camera with a 6-inch (152mm) focal length lens and from airplanes flying at 20,000 feet (6 km). Camera tilt angle is controlled and guaranteed less than 4°. Film-negative size is 9 by 9-inch, yielding photos of areas a bit more than 5 miles on a side. The camera optics and film have spatial resolution sufficient to resolve objects 1 to 2 meters in size. Digital images were produced scanning photos 1:1 from the negative films at 8-bit, 600 dots per inch, a sample definition of about 1.7 meters per pixel, to preserve information.

The SPIN-2 imagery comes from the former Soviet Kosmos Program, now available from the association of companies Sovinforsputnik (Russia), Aerial Images Inc. and Microsoft [Ref. 9]. The images were taken from a 1000mm focal length KVR1000 camera on satellites orbiting at 220 km, providing 2-meter spatial resolution. The images are digitized and distributed orthorectified and geo-referenced to precise accuracy, of 8 bits and 1.56 meter per pixel. [Ref. 10].

THIS PAGE INTENTIONALLY LEFT BLANK

III. COMPUTER VISION METHODS

A. OVERVIEW

Computer-vision methods try to recognize objects and infer facts from digital images. Produced by either direct digital capture or by scanning photographic film, the images can then be represented by bidimensional arrays of pixels, each pixel containing a number that describes a luminance value. The images are usually projections of the tridimensional world, from the perspective of the camera.

Computer-vision models and algorithms are frequently based on presumed characteristics of the human vision. One of them is the hierarchical organization. This means that recognition of complex shapes is obtained by first recognizing elementary patterns, then recognizing more complex patterns based on their positional relationships.

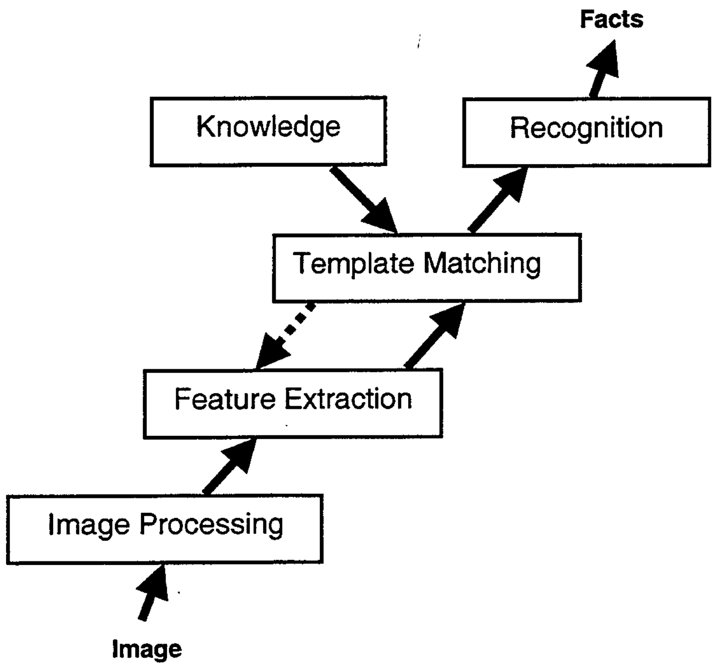


Figure 1: A hierarchical model for computer vision: from image data to facts.

B. IMAGE PROCESSING

Image enhancement, edge detection, and thresholding are commonly applied on digital images as a first step in extracting information. Linear and non-linear filtering are extensively used in these steps. An often-used technique is the neighborhood-based processing. Each pixel P(i, j) of the image has a set of neighbor pixels called *structuring element* or *neighborhood*, given by a selection function $\mathcal{N}$. A new array Q is created where Q(i, j) is assigned to f($\mathcal{N}$ (P(i, j))) for every **i** and **j**, for some mapping function **f** , usually one easily computable.

The filter, given by the composite function $f \circ \mathcal{N}$ is then an operator over the image space, returning a new image from the original one. This allows the cascading of filters until the information of interest is emphasized.

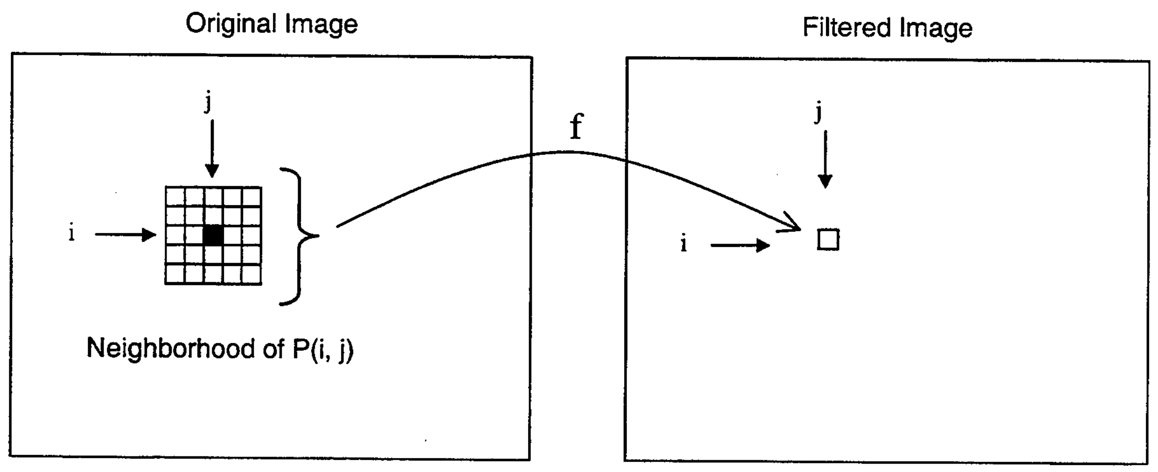


Figure 2: Image processing based on neighborhood mappings.

C. FEATURE EXTRACTION

Features are geometric patterns in images. The most important and basic of them is the line segment, because its occurrence in aerial photographs is often associated with human-made structures. Interesting higher-complexity features, like right angles, parallel line segments, and rectangles can be defined with straight-line segments.

Parametric transforms are standard methods of accomplishing feature extraction. They map an image from a *primary domain* into a transformed space where it is easier to identify geometric features. The transformed space, which is also called a *parametric space* or *transformed domain,* is often a bidimensional space that can be displayed as an image as well. Two commonly used parametric transforms for the detection of straight lines are the Hough and the Radon Transforms. We chose the second due to its fast discrete implementation with the Fast Fourier Transform.

D. THE RADON TRANSFORM

The Radon Transform [Ref. 11] works similarly to the easier-to-understand Hough Transform. It was devised in 1917 and it remained almost unnoticed until it became largely used in tomographic reconstruction. It was originally defined for continuous functions. Its main properties are:

(i) It maps Cartesian into polar pixel coordinates and replaces the complex search for aligned topologically connected clusters of pixels by the search for relative maxima in the transformed image.

(ii) Each pixel of the transformed domain will be associated with a unique line in the original image. The larger the transformed image, the finer will be the granularity in the representation of lines in the original image.

(iii) Distinguishing collinear line segments in the Radon Transform is impossible since their mappings coincide. Hence extraction of line segments requires additional processing after the transform.

For a continuous image domain $a:\Re^2 \rightarrow \Re$, the Radon transform is:

$$\hat{a}(\rho, \phi) = \mathrm{Radon}_{[a]}(\rho, \phi) = \int\int f(x, y)\, \delta(x \cos(\phi) + y \sin(\phi) - \rho)\, dx\, dy, \qquad (1)$$

where $\delta:\Re \rightarrow \Re$ is the Dirac Delta function.

The angular coordinate ϕ ranges in the interval $[-\pi/2, \pi/2]$, and the distance ρ to the center C can assume negative values according to the convention in Figure 3a.

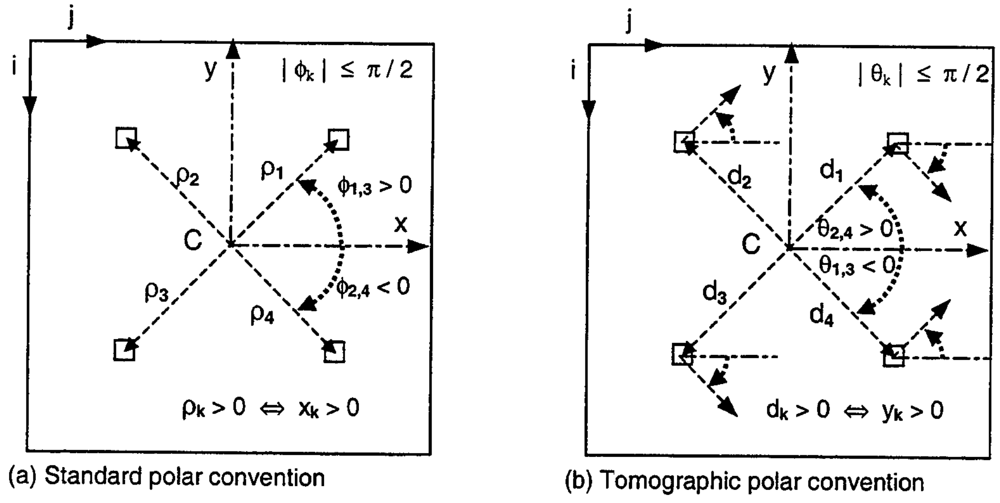


Figure 3: Polar coordinate conventions used in the Radon Transform.

The interpretation for $\text{Radon}_{[a]}(\rho, \phi)$ in equation (1), when "a" is a binary image, is the total length of the intersections of "on" areas of "a" and the line path $L_{\rho\phi}$ given by the equation $x \cos(\phi) + y \sin(\phi) - \rho = 0$. In the discrete approximation of the Radon transform (DRT), this is roughly proportional to the number of "on" pixels crossed along this line.

The DRT can be efficiently computed by the Fast Fourier Transform (FFT) algorithm because it can be expressed as the inverse one-dimensional transform in the radial variable ρ of the bidimensional Fourier Transform of $f(x,y)$ [Ref. 12]. The DRT is commonly implemented using the polar convention (θ, d) as in Figure 3b, θ being the "direction of inspection" associated with line $L_{\rho\phi}$. The discrete evaluation of the Radon Transform, R(m, n), can be displayed as an image called *sinogram*.

IV. MODEL DESCRIPTION

We present now our approach to the analysis of man-made structures in aerial photographs, which follows the same guidelines proposed by Ablameyko and Lagunovsky [Ref. 13, 14], but where the search for shapes is not limited to rectangles. Connectivity analysis of a graph derived from the line-segment representation of the image is performed aiming at the recognition of more general building-like contours.

For easy visualization of the process through its phases, intermediate results refer to the same image, cropped from a 1993 NAPP photograph (NAPP Roll# 6354 Frame# 253, of June 12, 1993) of Monterey, California. This image, in Figure 4 below shows an area of roughly 0.28 km^2 centered at latitude 36.60237 N and longitude 121.86555 W; it was digitized at approximately 1.78 pixel/m, 8 bit/pixel, yielding a 256 gray-tone, 340x260 pixel image.

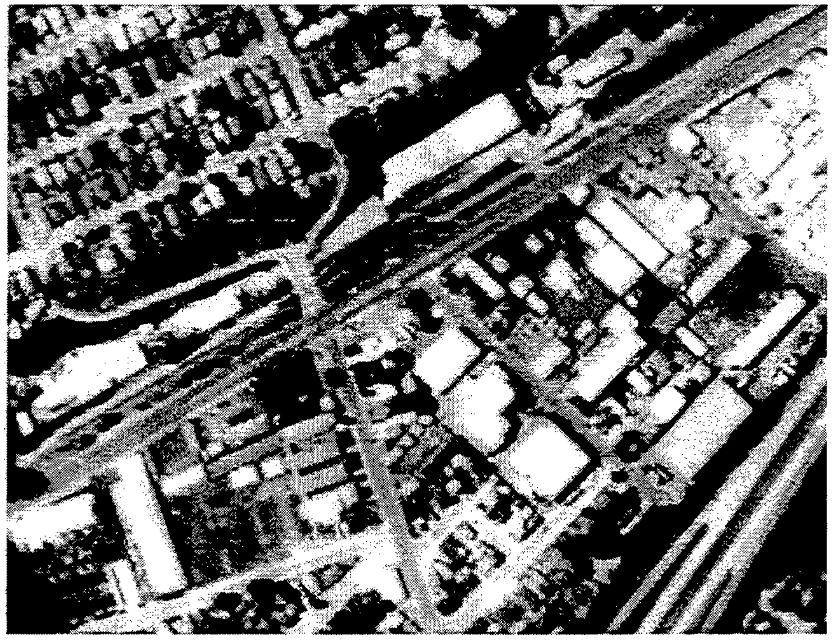


Figure 4: Sample NAPP aerial photograph of Monterey, California, USA.

A. EDGE DETECTION

1. Basics

Edge detection is the process of locating the main boundaries in a given image. Boundaries will exist between any two regions of the image exhibiting different average properties of color, luminance or texture. For gray-tone images, luminance differences are paramount.

The general approach for edge detection in gray-tone images is to first compute the gradient magnitude of the image, and then find the strongest edge pixels by thresholding. This operation produces a binary image of the same size of the original one, with pixels set to one where the gradient magnitude was high enough, to indicate a plausible boundary pixel, and set to zero otherwise.

2. The Canny Algorithm for Edge Detection

Use of a single threshold for detecting edge pixels may cause many important edge misses, if the threshold value is taken too high; if it is too low, some uninteresting boundaries or even noise in the image may cause the erroneous detection of edges. So Canny proposed [Ref. 15] an algorithm that introduces hysteresis in the thresholding. Edge strengths of topologically connected pixels are reenforced by strong gradient values of neighbor pixels. That improves the chances that major portions of boundary curves will be detected as a contiguous edge. The steps of Canny's algorithm are:

(i) A Gaussian smoothing filter is applied to the original image.

(ii) The gradient magnitude and direction is estimated at each pixel by directional differentiation operators.

(iii) Edge candidate pixels are located by computing the points of relative maxima in the gradient magnitude along the gradient direction, an operation referred as *non-maximal suppression.*

(iv) Edge candidates are inspected by a topological threshold rule. All pixels with gradient magnitude above the high threshold η_H are assumed to be edge pixels. The "edge influence" is then propagated by recursively making every pixel with gradient magnitude higher than a low threshold η_L also an edge pixel provided there is some previously found edge pixel in its immediate 8-neighborhood (see Figure 5).

$p_{i-1, j-1}$	$p_{i-1, j}$	$p_{i-1, j+1}$
$p_{i, j-1}$	p_{ij}	$p_{i, j+1}$
$p_{i+1, j-1}$	$p_{i+1, j}$	$p_{i+1, j-1}$

Figure 5: Representation of the 8-connected neighborhood of a pixel p_{ij}.

Canny's method, one the most successful general edge-detection algorithms [Ref. 16], was chosen for this study after excelling in tests we did against multilevel thresholding edge detectors. It meets some optimality criteria concerning non-spurious edge detection, accuracy on the edge location and avoidance of double-edge detection, as shown in Canny's original paper of 1986 [Ref. 15] and textbooks on image processing [Ref. 17].

Application of Canny's method to Figure 4 gives Figure 6. Black appears where edge pixels were detected. Although detail is lost in this operation, the major part of the building and road contours is preserved, so that a trained human observer would be able to detect and recognize them. Generally speaking, any intermediate representation of visual information should still be meaningful to the eye of an expert.

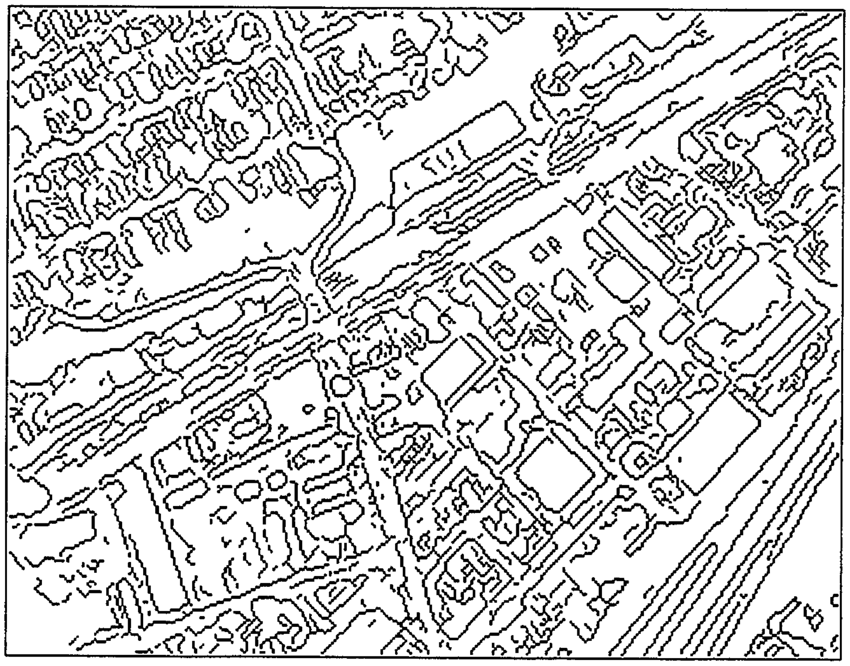


Figure 6: Binary image obtained with Canny's edge detector.

B. EDGE ENHANCING BY MORPHOLOGICAL FILTERING

To reduce the edge-coupling interference, morphologic filtering is used to rupture the edge segments before the extraction of primitive line segments described below. Good candidates for rupture points are corners of right angles and convergence points as forks, joints, and crosses.

Morphologic filtering is performed by mapping small squared pixel neighborhoods using fast lookup table implementation. Specifically, we used 3x3 neighborhoods and a lookup table of length $2^{3x3} = 512$. Figure 7 shows all the considered cases for rupture points. For such points the corresponding pixel in a special binary image **K** (same size of **E**) is set to one, with K(i, j) = 0 elsewhere. Figure 8 shows the rupture points for the edge image in Figure 6. It can be noticed that rounded corners were missed, but many others corners were detected.

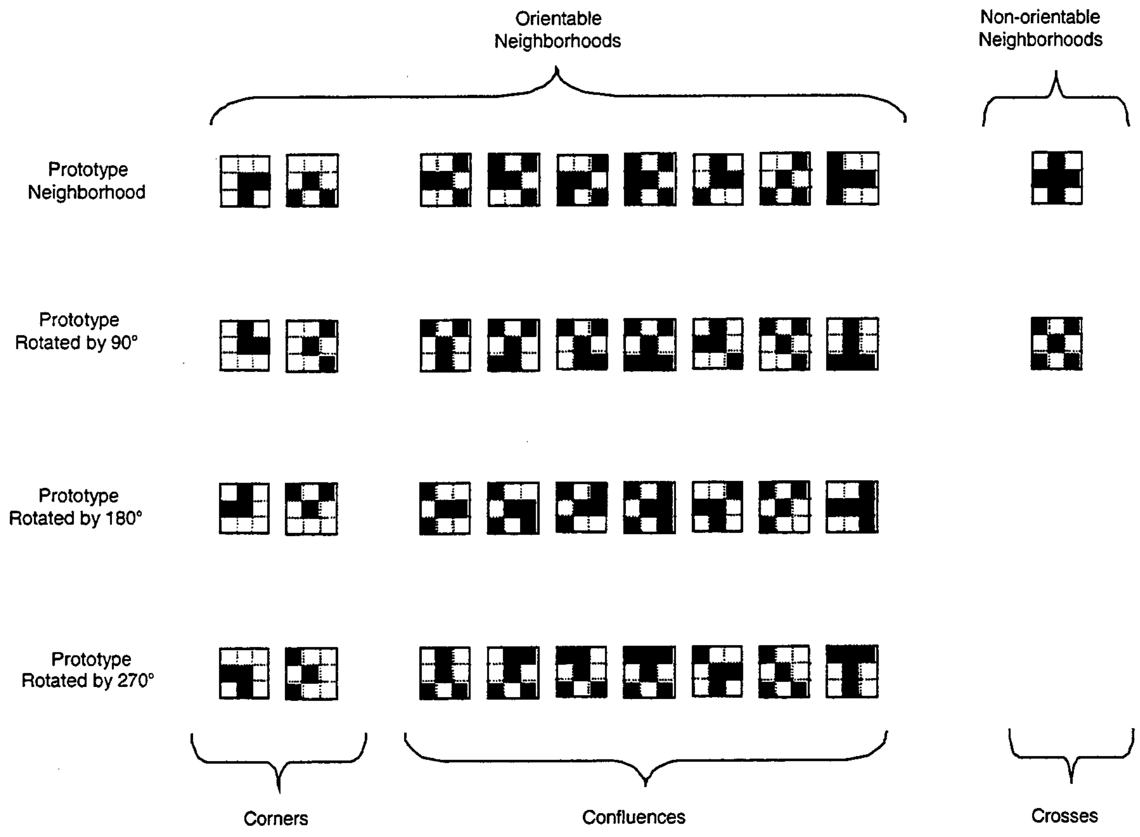


Figure 7: 3x3 neighborhoods for detection of prospective rupture points in a binary edge image.

The image segmentation is then performed on the difference image E – K , not on the original image E. K is used again in the primitive line segment extraction phase (IV.C.2): After segmentation, bounding boxes are enlarged by a one pixel-wide envelope and segments recomputed including points in K, to preserve more of edge information.

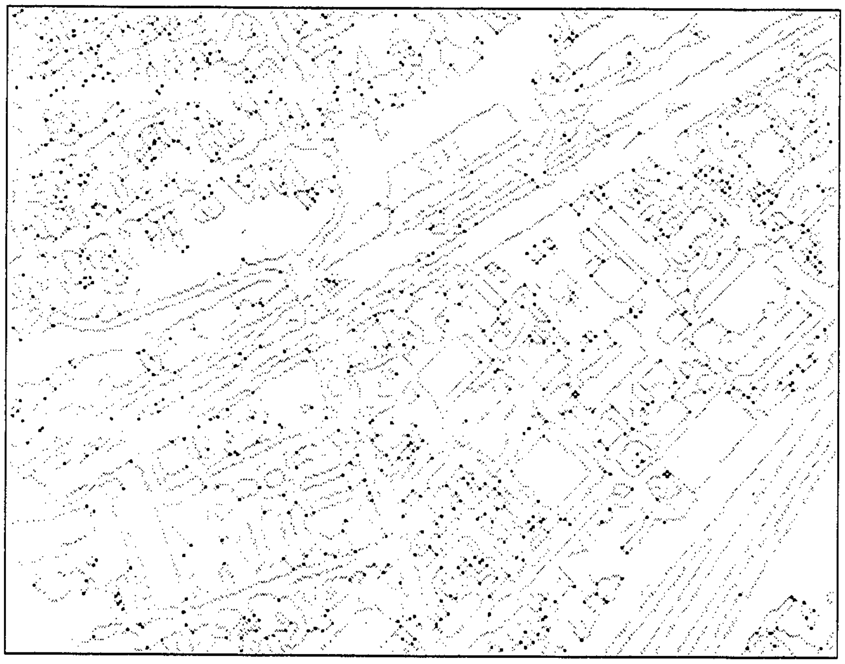


Figure 8: The calculated rupture points (black dots) superimposed on the edge image.

C. PRIMITIVE LINE EXTRACTION

1. Basics

The binary image produced by edge detection contains a set of edge pixels. The next task is to produce a list of *line segments* from this binary image, because these are key in identifying man-made structures. Primitive-line extraction is the first level of symbolic representation of the image and feature extraction. The features, called *primitive line segments* (PL), are straight-line segments along region boundaries in the original image.

Line segments are defined by a pair of points (P_1, P_2). In the two-dimensional space of images, each point location can be specified by two numbers, so each segment is defined by four numbers. Although the resolution of the original image is limited to the pixel size, the PL representation can use floating point numbers. The precision in segments location is potentially better than the pixel size because of the large number of pixels used to determine each one.

Since most of the analysis operations use the angular information, line segment data redundantly stores its inclination angle θ, the coordinates of the base point B (projection of the center of the image onto the line), and the distance **d** to the center of the image:

$$PL = (\theta, d, B_i, B_j, P_{1i}, P_{1j}, P_{2i}, P_{2j}) \tag{2}$$

Figure 9 illustrates this. (θ, d) are the polar coordinates with respect to the center C of the image, d being positive when the base point is above the center (following convention of Figure 3b). Pixel locations in the image, on the other hand, are expressed in the Cartesian coordinate system (i, j), with the origin at the upper-left corner of the image.

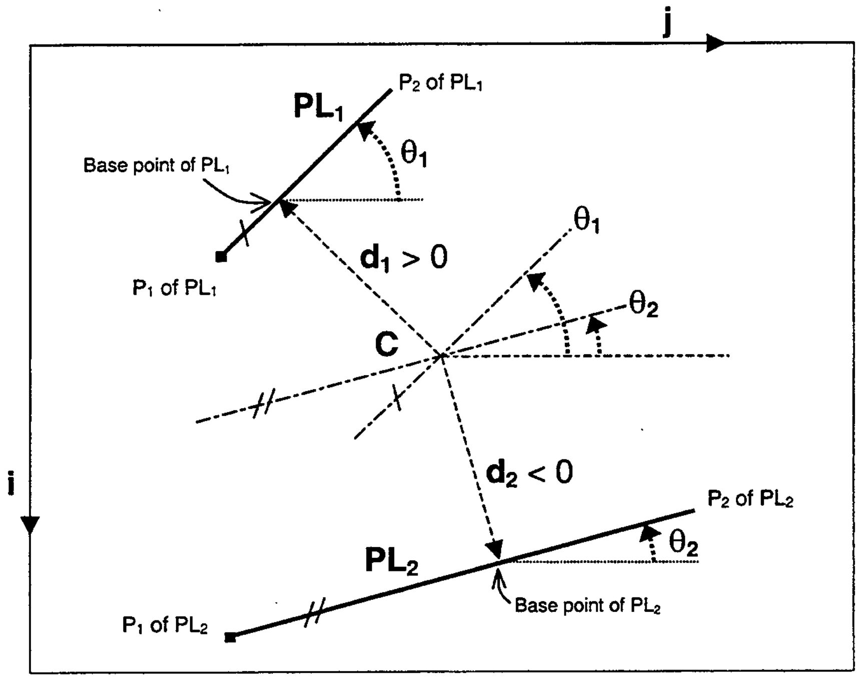


Figure 9: Redundant coordinate system used to represent primitive lines.

2. Line Extraction with the Radon Transform

The first step in line extraction is the segmentation of the binary edge image "E" into sets of connected pixels, using a 8-cell neighborhood to define pixel connectivity. The partition of edge pixels ξ into the set $\{\xi_i\}$ is accomplished such that equation (3) holds. Figure 10 exemplifies this operation for an edge image containing $n_E = 3$ segments.

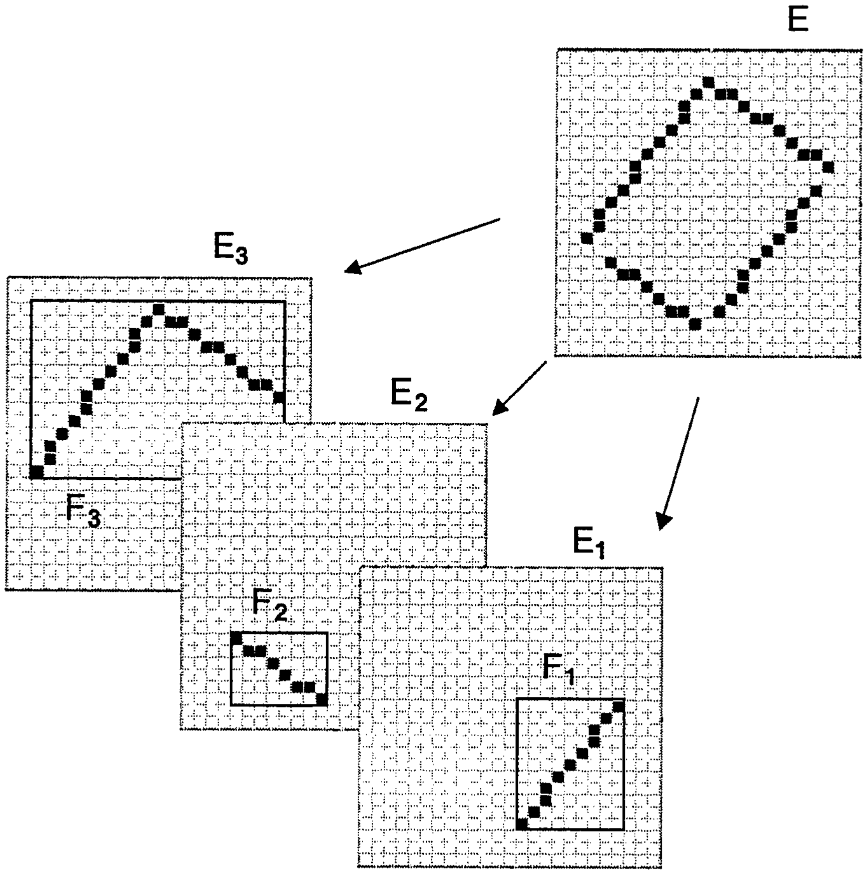


Figure 10: Example illustrating the segmentation of a binary edge image.

$$\xi = \bigcup_{i=1}^{n_E} \xi_i \,, \text{ with } \bigcap_{i=1}^{n_E} \xi_i = \varnothing, \tag{3}$$

where $1 \leq i \leq n_E$, n_E = number of segments in E.

Extracting lines from an edge binary image is accomplished by the Radon Transform (presented in III.D). The Transform is applied to the bounding box F_i around each pixel set ξ_i to save further computation time, resulting in a coefficient array R_i that can also be plotted as an image. Figure 11 shows the plot of the Radon Transform for edge image E_3. The relative maxima in the transform corresponds to possible lines in the binary image. In Figure 11, there are two relative maxima, marked with crossing lines, the one at $(\theta_{3,1}, d_{3,1}) = (53^\circ, 6)$ and one at $(\theta_{3,2}, d_{3,2}) = (-34^\circ, 8)$.

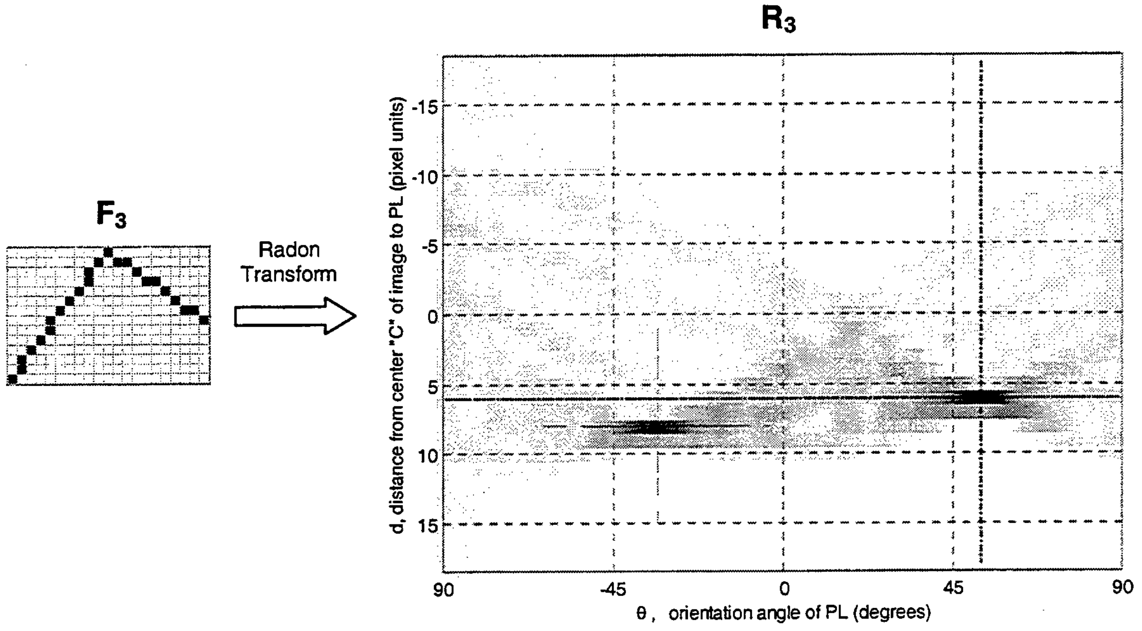


Figure 11: Plotting of R_3, the Radon Transform for edge image E_3.

3. Determining Line Segment Endpoints

The Radon Transform maxima give only lines through potential line segments; they do not give the endpoints of the line segments. This second task is accomplished by first masking out all the pixels in F_i not on a 3-pixel-wide linear band centered on the support line. Because this masking may break the connectivity of the pixels in the linear band, a new segmentation is done with these pixels. For each of the resulting clusters, a primitive line segment can be fit using a least-squares estimate based on the projections on the support line of those pixels at extreme xy coordinates.

In order to reduce inter-edge influence, after extracting all possible primitive line segments in one direction, instead of continuing and proceeding with the next relative maximum, the Radon Transform is recomputed on the remaining pixels. To guarantee best accuracy, only the line associated with the maximum Radon coefficient is inspected at each iteration. This process is recursively repeated until the maximum Radon coefficient reaches a minimum value corresponding to the integration of two pixels. At this point, the original edge image is exhausted and no more useful pixels remain unconverted to primitive line segments.

This transform recomputation slows the algorithm, but not directly in the proportion of the number of iterations, because the order of the Radon transform at each iteration becomes smaller too. Its benefit is a more accurate conversion of edge pixels into primitive line segments in low-definition images once mutual edge-coupling interference is strongly reduced. A further enhancement in the line extraction is a line-fitting algorithm based on the minimization of squared distances from pixels in a cluster to the modeling line (see item IV.C.4 below).

4. Mean Square Error Line Segment Estimator

The Radon line extraction of the segmented edge image just described in IV.C.2, isolates clusters of aligned 8-connected pixels. For each of these clusters a line segment is computed by projecting the center of the end-pixels on the line that minimizes the sum of squared distances to them. If (d_i, θ_i) are the polar coordinates of each of the **c** pixels $\mathbf{P}_i$ in a cluster, for $L(d, \theta)$ a given line, it can be derived that (see Figure 12):

$$\varepsilon_i = d_i \cos(\theta - \theta_i) - d \qquad (4)$$

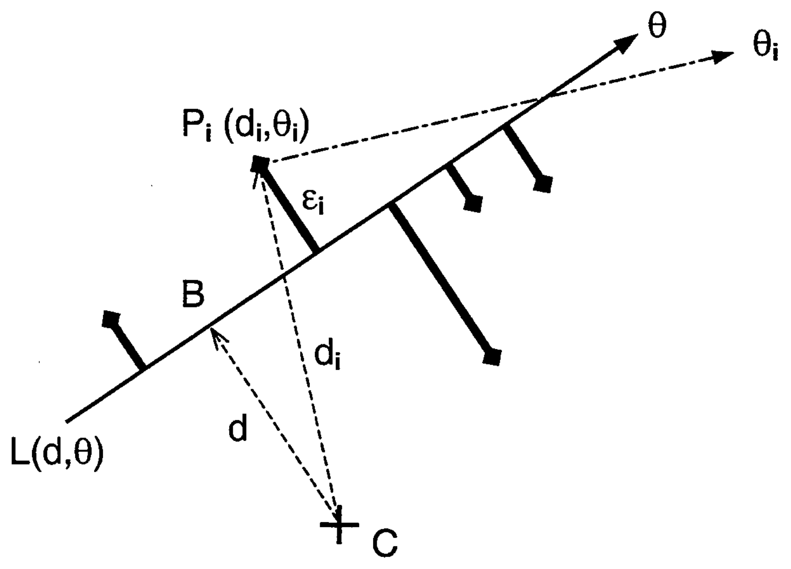


Figure 12: Treating distances ε_i from the center of pixels P_i to the fitting line L as errors to be minimized by least squares method.

$\Sigma\ \varepsilon_i^2$ has a quadratic form in **d**, and it always non-negative. Hence by making $\partial\Sigma\ \varepsilon_i^2/\partial d = 0$, the distance $\tilde{d}$ that minimizes $\Sigma\ \varepsilon_i^2$ for a fixed angle θ is:

$$\tilde{d} \ = \ \sum_{i=1}^{c} d_i \cos(\theta - \theta_i) \,/\, c \tag{5}$$

If $\tilde{\theta}$ is an angle that minimizes $\Sigma\ \varepsilon_i^2$, then $\partial\Sigma\ \varepsilon_i^2/\partial\theta = 0$ and we derive:

$$\sum_{i=1}^{c} d_i \cos(\tilde{\theta} - \theta_i) \sin(\tilde{\theta} - \theta_i) \ = \ d \sum_{i=1}^{c} \sin(\tilde{\theta} - \theta_i) \tag{6}$$

Simultaneously searching in θ and d for minimum in $\Sigma\ \varepsilon_i^2$ requires making $\theta = \tilde{\theta}$ in equation (5) and $d = \tilde{d}$ in equation (6). Solving the system of equations formed:

$$\sum_{i=1}^{c} c\ d_i \cos(\tilde{\theta} - \theta_i) \sin(\tilde{\theta} - \theta_i) \ = \ \sum_{i=1}^{c} d_i \cos(\tilde{\theta} - \theta_i) \,.\, \sum_{j=1}^{c} \sin(\tilde{\theta} - \theta_j) \tag{7}$$

Developing (7) gives two possible values for $\tilde{\theta}$:

$$\tilde{\theta} = \text{arc tan} \left(\frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}\right), \tag{8}$$

where:

$$\alpha = \sum_{i=1}^{c} c\, d_i \sin(\theta_i) \cos(\theta_i) - \sum_{i=1}^{c} \sum_{j=1}^{c} d_i \sin(\theta_i) \cos(\theta_j) \tag{9}$$

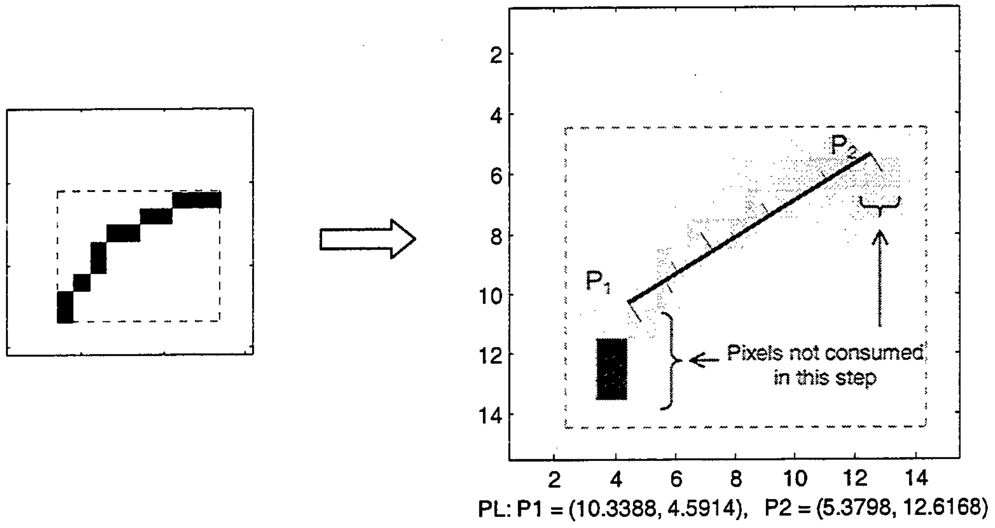
$$\beta = \sum_{i=1}^{c} c\, d_i (\cos^2(\theta_i) - \sin^2(\theta_i)) - \sum_{i=1}^{c} \sum_{j=1}^{c} d_i (\cos(\theta_i) \cos(\theta_j) - \sin(\theta_i) \sin(\theta_j)) \tag{10}$$

$$\gamma = \sum_{i=1}^{c} \sum_{j=1}^{c} d_i \cos(\theta_i) \sin(\theta_j) - \sum_{i=1}^{c} c\, d_i \sin(\theta_i) \cos(\theta_i) \tag{11}$$

For each trial $\tilde{\theta}$, a value for $\tilde{d}$ is computed by equation (5). The fitting line $L(\tilde{d}, \tilde{\theta})$ is found by selecting the $\tilde{\theta}$ and $\tilde{d}$ that yield minimum computed $\Sigma\, \varepsilon_i^2$ through equation (4). If all pixels were either vertically or horizontally aligned with the reference point C, quantity α in the above equations would be zero, leaving $\tilde{\theta}$ undetermined. To prevent that, instead of using the actual center of the image, whose coordinates are always multiples of 0.5, the polar origin for this computation is momentarily displaced by a number that is not a multiple of 0.5. After $\tilde{\theta}$ is computed, the polar origin is brought back to the center of the image, so that the correct parameters for the line segment are extracted. Situations where a real number for the angle $\tilde{\theta}$ does not exist ($\beta^2 - 4\alpha\gamma < 0$) in practice will not occur because the Radon Transform masking imposes some a priori alignment to the pixels.

The missing link to obtain the parametric representation of the cluster of pixels is the estimation of the endpoints of the line segment on the support line $L(\tilde{d}, \tilde{\theta})$. They are computed by taking those projections of $\mathbf{P}_i$ on L with minimum and maximum (x, y) coordinates.

Figure 13 illustrates the geometry of the line fitting process. Along an edge, the pixels not selected by the Radon maximum-coefficient mask are left for the next iteration of the algorithm (black pixels, on the right). The pixels that were fit by the line segment are removed from consideration for subsequent edges, except the end pixels, which are spared for newer iterations. This last action helps line segments extracted from contiguous edges to have endpoints closer to each other, help subsequent building-contour tracing (see IV.F).



P₂
P₁
Pixels not consumed in this step
PL: P1 = (10.3388, 4.5914), P2 = (5.3798, 12.6168)

Figure 13: Primitive line segment extraction with Radon masking (light gray) and line fitting of the enclosed pixels (dark gray).

5. The Overall Effect of the PL Extraction Phase

The final set of primitive line segments visually resembles the edge binary image when plotted. For the example image of Figure 6, the plot of its 1858 primitive line segments can be seen in Figure 14. Some isolated groups of pixels were "lost", but this is actually an additional convenient simplification of the original image.

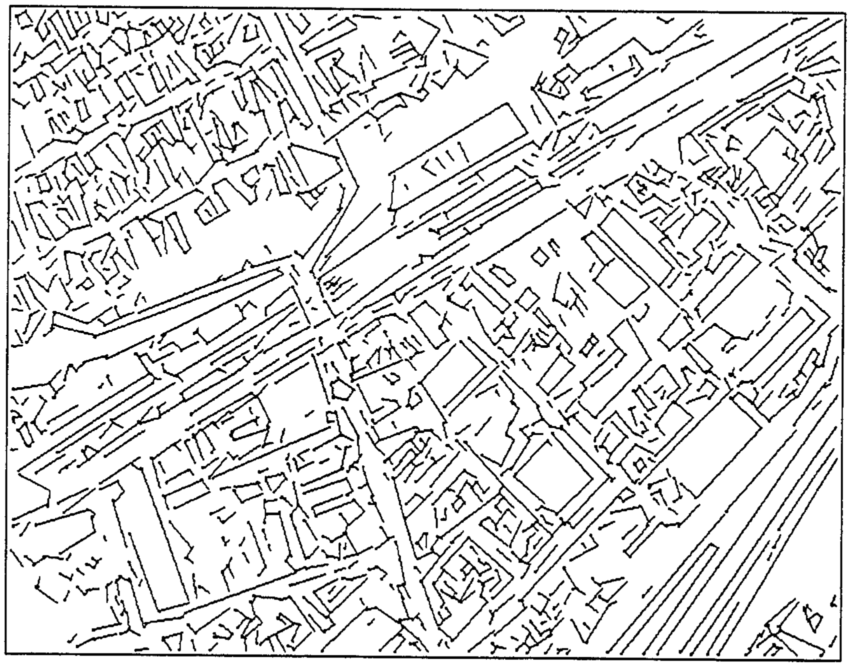


Figure 14: Plotting of the final line segments extracted from the image in Figure 6.

This resemblance is in fact so high that may confuse the observer. Enlarging corresponding regions of both images (Figure 15), the parametric nature of the primitive lines becomes apparent.

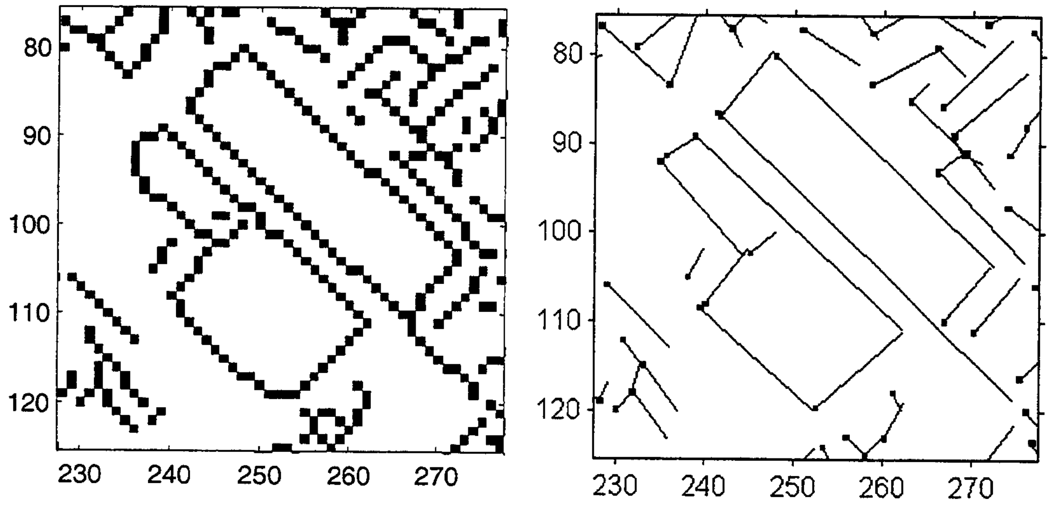


Figure 15: Detail comparing edge pixels (left) with computed line segments (right).

D. CLUSTERING OF PRIMITIVE LINE SEGMENTS

The typical number of primitive line segments extracted from our test images was one per 50 pixels. Images of modest size can produce line-segment descriptions with thousands of lines. The algorithms for higher-level feature extraction, that take line segments as input, search for combinations of line segments that will match some positional relationships. This can be very expensive computationally ($\mathcal{O}(n^3)$ or higher).

To improve computational speed, the initial line-segment set is broken into smaller-sized sets by a relatively fast clustering algorithm. Since urban areas and buildings are the main objects of interest, partition sets should be constituted by lines within blocks, streets being the boundaries. This partition should satisfy the following constraints:

(i) The line segments of any real-world object should be in the same partition.

(ii) Partitions should correspond to contiguous areas in the original image.

If an algorithm based on the line space search is $\mathcal{O}(n^s)$, $s > 1$, pre-clustering the primitive lines into **k** groups of **m** lines will result in reduction of the complexity order of the problem. The new computational time will be in the order of:

$$\mathcal{O}(k\,\mathcal{O}(m^s)) = \mathcal{O}(\frac{n}{m}\mathcal{O}(m^s)) = \mathcal{O}(n\,m^{s-1}) \tag{12}$$

If the number **m** of lines in each cluster could be limited, the factor m^{s-1} would be modeled as a constant, and the application of the higher-level algorithms would be $\mathcal{O}(n)$, yet the large constant involved m^{s-1} could make this a "slow" $\mathcal{O}(n)$ algorithm.

$$\mathcal{O}(n\,m^{s-1}) = m^{s-1}\,\mathcal{O}(n) = \mathcal{O}(n) \tag{13}$$

The algorithm used to cluster the line segments has two steps: First, an undirected graph G is calculated for endpoints of the primitive line segments. The vertices of the graph G correspond to line segments and an edge is created if either:

(i) The two line segments touch at some of their endpoints, within the resolution of the image; or

(ii) The line segments are perpendicular to each other, within an angle of $\pi/8$ radians, and their touching endpoints are closer than the geometric mean of their lengths.

Second, clusters of connected segments are found by looking for connected subgraphs of G. Since single-segment clusters are uninteresting, we discard segments that remained unclustered.

The criterion in (i) is intuitive. The criterion in (ii) was introduced because perpendicular lines tend to be related in man-made structures. It has the desirable property of being scale-independent. The criterion of minimum length was also tried, but did not yield as good results as the geometric mean.

Application of the algorithm to the primitive lines in Figure 14 results in Figure 16, where the partitions obtained are coded in different colors. In this example, of 1858 original line segments, 5% were discarded after being unclustered; the remaining were clustered into 71 sets with the maximum of 351 and the average of 25 line segments per cluster. In images tested, the clusters appeared more dependent on the nature of urban area than either the size of image or the total number of lines extracted.

This gives some support to the hypothesis upon which equation (13) was derived. Since the algorithm that produces the connection matrix G runs in time of order $\mathcal{O}(n^2)$, this might also be the order of the global line segment analysis, provided the order of higher-level feature extraction in the following phases of the analysis is kept at polynomial order.

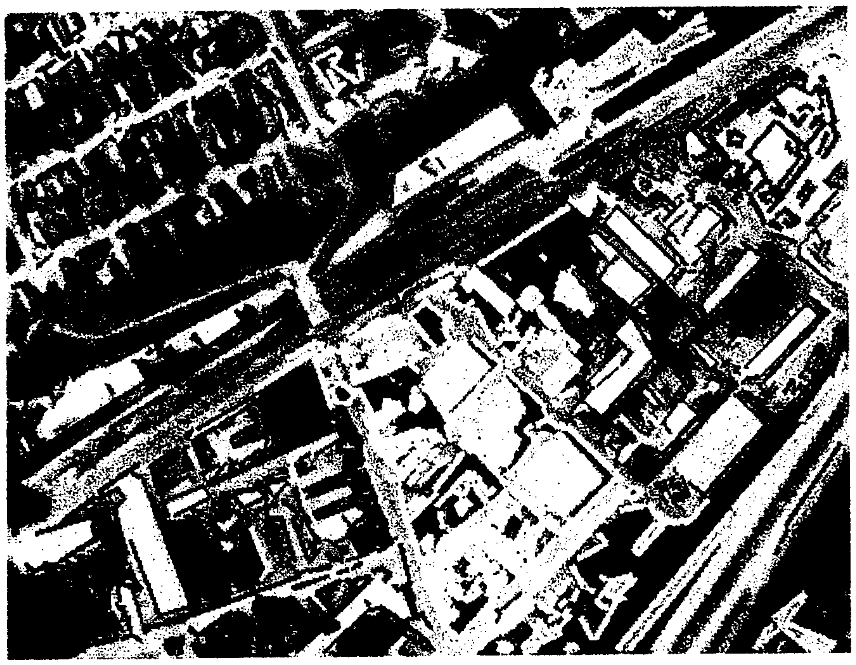


Figure 16: Clustering of primitive line segments, plotted with assorted colors, for the image in Figure 4.

E. MERGING OF PRIMITIVE LINE SEGMENTS

The aerial visibility of edges of buildings and other objects may be obstructed by trees or shade, or may be degraded due to lack of contrast between the object and the background. This may cause a single physical edge to be segmented into several collinear line segments. To facilitate the detection of the polygonal shapes that characterize man-made structures like buildings, we merge close and approximately collinear primitive line segments.

1. The Merge Procedure

Merging of line segments is accomplished by:

(i) Find pairs (L_1, L_2) of line segments which are oriented in approximately the same direction, within a maximum heading deviation of $\Delta\theta_{max}$, and whose distances to the center of the image differ at most by Δd_{max}.

(ii) Use a *relative position criterion* to eliminate pairs in a side-by-side configuration, as exemplified in Figure 17(b). Require that the maximum distance from an endpoint of L_1 to an endpoint of L_2 be attained at the points that are opposite to those where the minimum distance is, as in Figure 17(a).

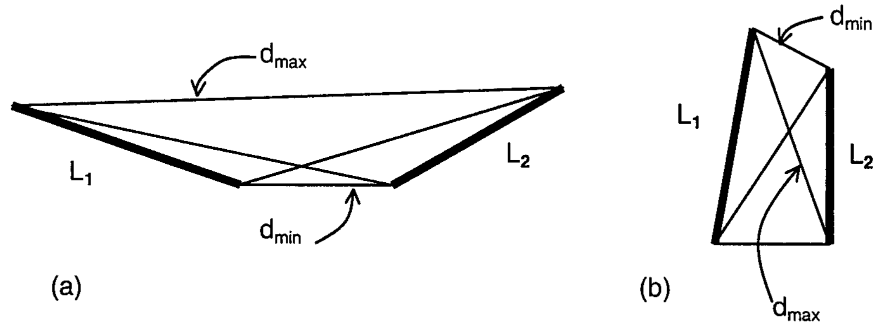


Figure 17: Examples of favorable (a) and unfavorable (b) relative positions for merge candidate line segments L_1 and L_2.

(iii) Require also that no other line segment have endpoints lying near the endpoints of the candidate segments, to prevent the suppression of possible corners (see Figure 18).

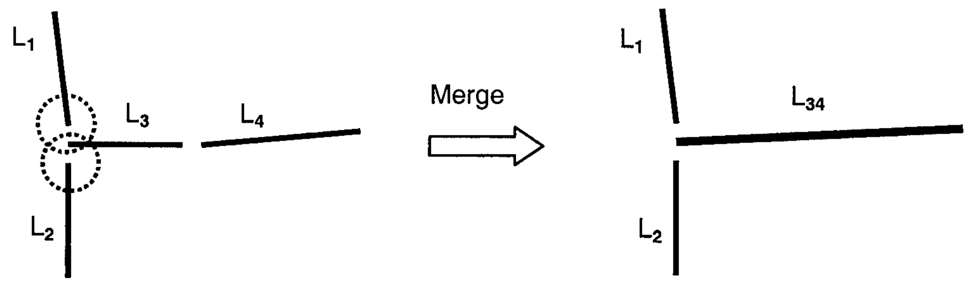


Figure 18: The presence of L_3 inhibits the merging of aligned segments L_1 and L_2, thus preserving the junction of L_1, L_2 and L_3; L_3 and L_4, on the other hand, can be merged.

(iv) Eliminate segment pairs failing to satisfy a *proximity criterion*: The minimum distance between endpoints of two candidate line segments should be less than the smaller length of the two line segments. In Figure 17, we must have:

$$d_{min} < \min\{ |L_1|, |L_2| \}, \tag{14}$$

$$\text{where } d_{min} = \min\{d_{ij} \mid i \text{ endpoint of } L_1, \ j \text{ endpoint of } L_2\} \tag{15}$$

(v) Eliminate pairs of segments failing an *alignment criterion.* Let $\mathbf{h}_{ijk}$ be the oriented distance (projection) from endpoint P_{ki} of line segment **k** to the support line of line segment **j**:

$$h_{ijk} = \text{distance}(P_{ki}, L_j) \tag{16}$$

Alignment is met by requiring that the angle subtended by rays through the endpoints of one line segment and rooted in one endpoint of the other segment be less than a constant ΔS_{max}. Additionally, to avoid the merging of parallel lines, the distance of endpoints of one segment to the other line should be less than the resolution distance $2\sqrt{2}$ pixels, if these endpoints are in the same side of the plane, with respect to the second line. In terms of the d_{ij} and the h_{ijk}, either of the following conditions should apply (see Figure 19) to the pair of line segments (L_1, L_2):

$$\left\{\begin{array}{l} \min\{\max\{|h_{112}/d_{11}|, |h_{212}/d_{12}|\}, \max\{|h_{112}/d_{21}|, |h_{212}/d_{22}|\}\} < \sin(\Delta S_{max}) \\ h_{112}\, h_{212} > 0 \Rightarrow \max\{|h_{112}|, |h_{212}|\} < 2\sqrt{2} \end{array}\right. \tag{17}$$

or

$$\left\{\begin{array}{l} \min\{\max\{|h_{121}/d_{11}|, |h_{221}/d_{21}|\}, \max\{|h_{121}/d_{12}|, |h_{221}/d_{22}|\}\} < \sin(\Delta S_{max}) \\ h_{121}\, h_{221} > 0 \Rightarrow \max\{|h_{121}|, |h_{221}|\} < 2\sqrt{2} \end{array}\right. \tag{18}$$

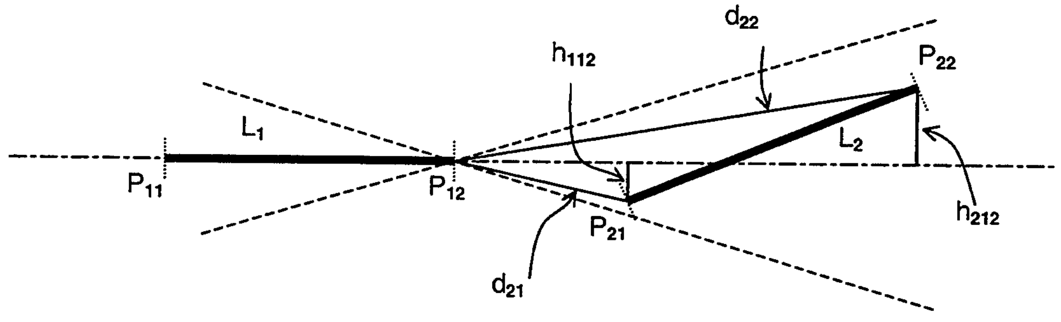


Figure 19: Alignment criterion for two line segments.

The resolution distance of $2\sqrt{2}$ is the maximum distance between any two points lying in the area of two neighbor pixels in 8-neighborhood. Such a condition is met when the points are in opposite corners of diagonally touching pixels. The angle ΔS_{max} was determined by trials and fixed at $\pi/36$ radians.

(vi) Finally, cluster the qualified merge candidate segments in fully connected sets. Then only merge a set if every pair within that set satisfies the merge criteria. This will assure a global alignment for the line segments, rejecting the merge of patterns like (b) and (c) in Figure 20.

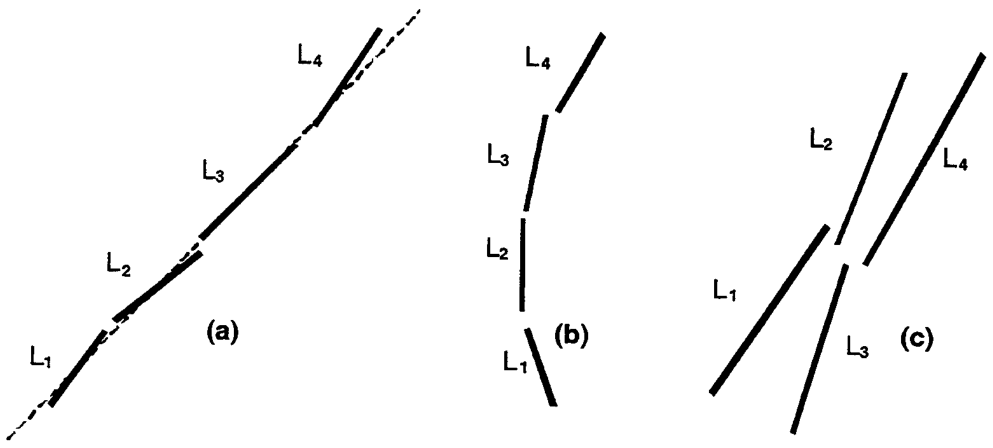


Figure 20: In all three clusters, the merging criteria are satisfied by any two consecutive line segments L_i, and no other pairs; however, only the cluster on the left exhibits a global alignment.

2. Overall Merge Effect

Merging the qualified primitive line segments does not affect the general appearance of the line segment plot, as seen in Figure 14 or Figure 16. But it does improve the edge extraction, as exemplified in the zoomed detail of Figure 21.

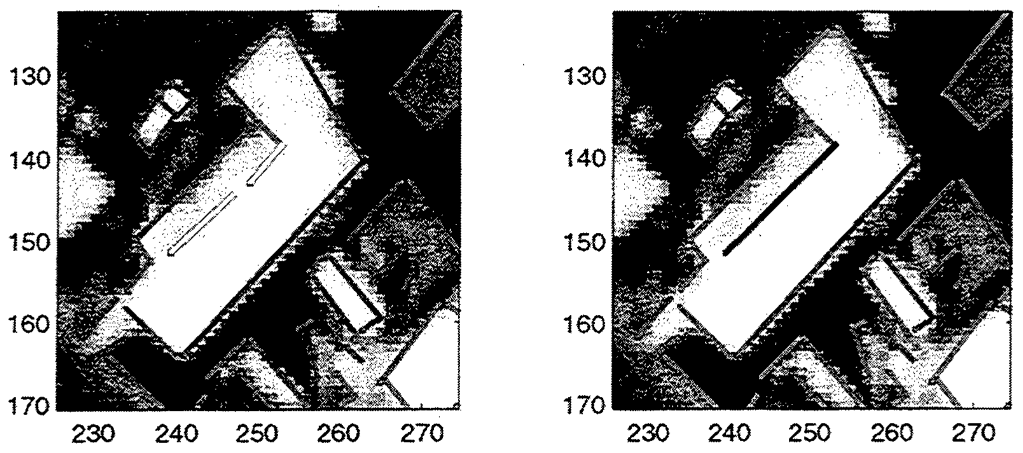


Figure 21: The effect of merging primitive line segments on the contour of a building: Before merging (left) and after merging (right).

F. SEARCH FOR BUILDINGS

Man-made constructions such as simple buildings and houses usually fit rectangular shapes well. But more complex buildings are better modeled by closed ortho-polygonal lines (polygons made of right angles). The detection of these can be accomplished by looking for cycles in a graph derived from the line segment representation of the image. In this graph, the vertices are endpoints of the line segments, and edges will be created where some useful geometric relationship exists between endpoints, similarly to the clustering phase described in IV.D.

1. Endpoints Graph Formulation

For each cluster C_k containing N_k primitive line segments extracted from the original image, we define a graph $\Gamma_k(V_k, G_k)$ where:

$V_k = \{v_1^{(k)}, v_2^{(k)}, v_3^{(k)}, \ldots, v_{2N_k}^{(k)}\}$ is the set of endpoints of the line segments in C_k;

and

$G_k = [g_{ij}^{(k)}]$, $1 \leq i, j \leq 2N_k$ is an edge connection matrix for vertices in V_k, defined according to the type of geometric relationship between v_i and v_j, as follows:

Type 1: $g_{ij} = 1 \Leftrightarrow v_i$ and v_j are endpoints of the same line segment;

Type 2: $g_{ij} = 2 \Leftrightarrow v_i$ is in the $\mathcal{N}^{21}$ neighborhood of v_j (see Figure 22) but v_i and v_j (as shown in Figure 22) are not endpoints of the same line segment (proximity criterion);

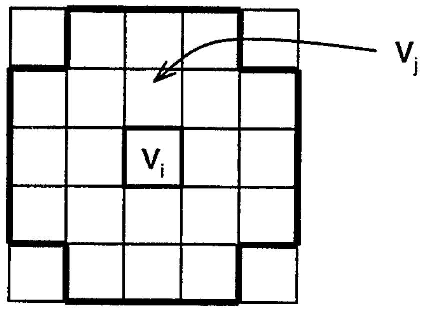


Figure 22: The $\mathcal{N}^{21}$ neighborhood as a criterion for connecting endpoints in the graph Γ_k.

Type 3: $g_{ij} = 3 \Leftrightarrow v_i$ and v_j are endpoints of line segments which are approximately perpendicular, and v_i and v_j are closer than the geometric mean of the lengths of the line segments, but not enough close to be $\mathcal{N}^{21}$-neighboors (situation shown in Figure 23);

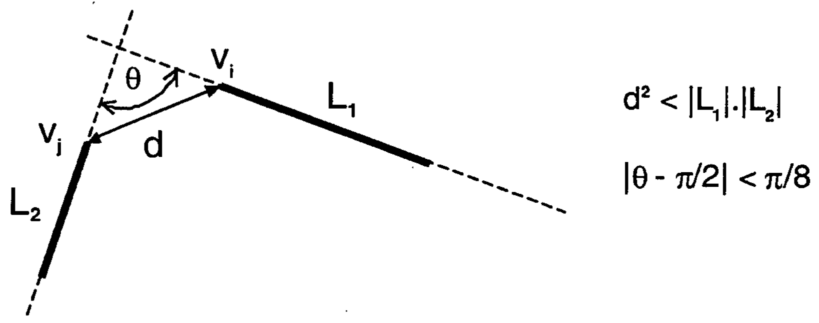
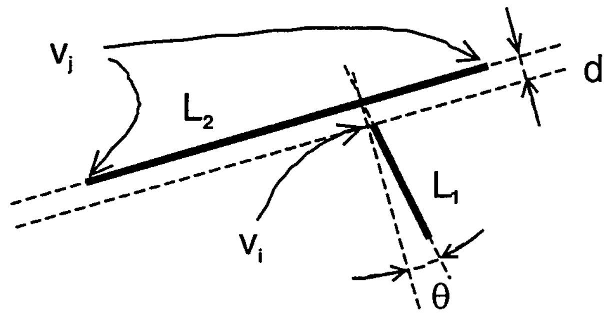


Figure 23: Connecting endpoints in corner position.

Type 4: $g_{ij} = 4 \Leftrightarrow v_i$ and v_j are endpoints of two approximately perpendicular line segments and the distance between v_i and the line segment containing v_j (or vice-versa) is less then $2\sqrt{2}$;

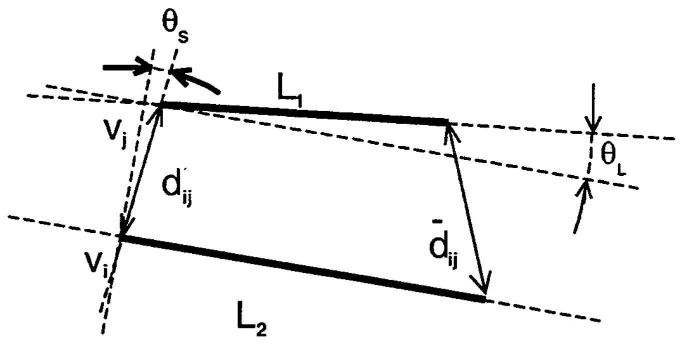


$d < 2\sqrt{2}$ pixel

$|\theta| < \pi/8$

Figure 24: Connecting endpoints in "T" position.

Type 5: $g_{ij} = 5 \Leftrightarrow v_i$ and v_j are endpoints of approximately parallel (within $\pi/18$ radians) line segments which are closer then the minimum of their lengths. Also, the line passing though v_i and v_j should be approximately perpendicular to the two line segments (within $\pi/18$ radians to the larger line segment).



$L_1 \le L_2$

$\max\{d_{ij}, \bar{d}_{ij}\} < \min\{|L_1|, |L_2|\}$

$\min\{|\theta_S|, |\theta_L|\} < \pi/18$

Figure 25: Connecting endpoints of parallel line segments in opposition.

The g_{ij} will be zero otherwise, representing that there is no relationship (and thus no edge in the graph) between endpoints v_i and v_j.

2. Finding Cycles in the Endpoints Graph

Building candidates are found by searching for cycles in the graphs Γ_k using depth-first search [Ref. 18]. The basic version of this algorithm travels along the edges until a closed path is found. When it is not possible to travel further, it backs up along the path until a vertex that offers a path option not tried before is found, and then it follows it. If backup continues until the initial vertex is found and no more path options remain, then the search for cycles starting at that node was unsuccessful. The search is restricted so that the current path should not contain the same edge twice.

Figure 26 shows an example set of line segments and the tree that was generated for finding a cycle from v_1; Figure 27 shows the implicit connection graph used. In this case, node v_1 is visited again at the sixth move, at a depth h=6 from the root node.

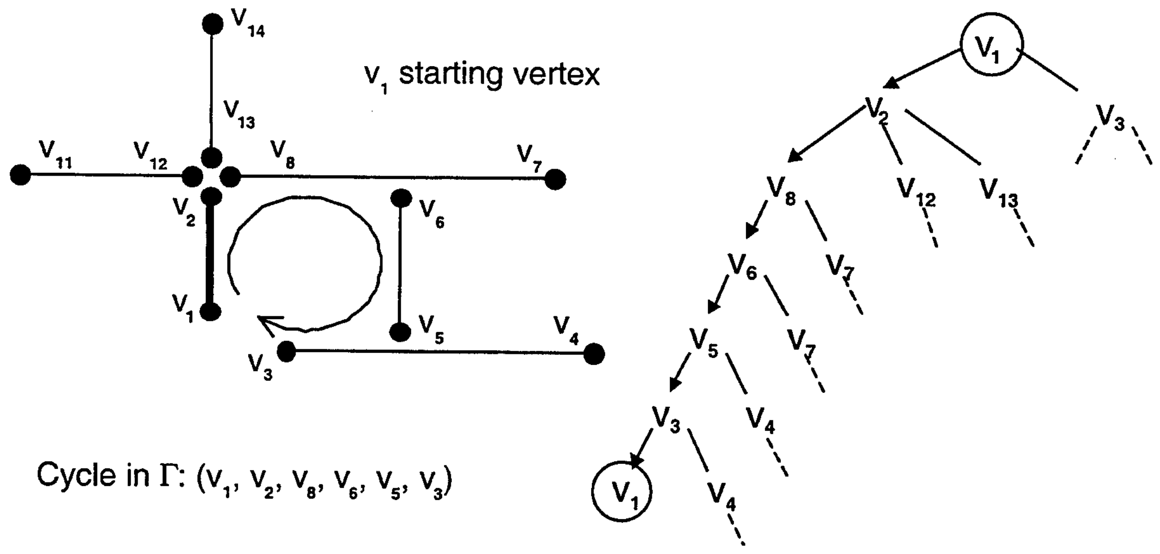


Figure 26: Example of standard depth-first search for cycles in graph Γ.

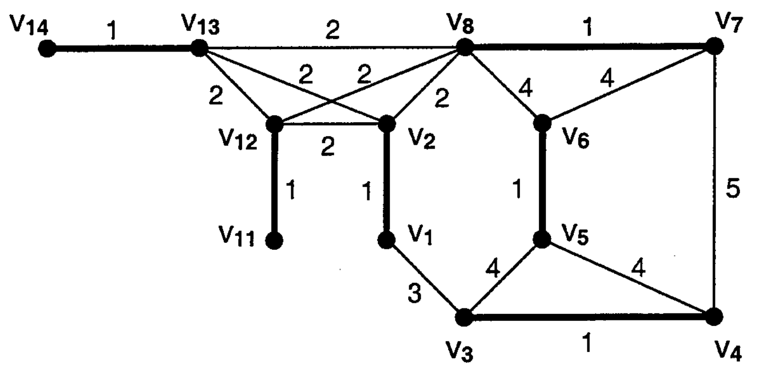


Figure 27: Graph Γ for the set of primitive line segments in Figure 26. Thick edges correspond to line segments. Thin edges signify other type of geometric relationship.

To guarantee that the searching path does not fold over itself or the cycle collapse into a double cycle, the visited nodes (except the root node) are prohibited to occur twice in the path. So are the nodes associated with rejected branching options at a given search pass. This is called the visited/prohibited rule. In Figure 27 for instance, after jumping from v_2 to v_8, nodes v_{12} and v_{13}, as well as node v_2 itself, become prohibited for that path.

Because of the visited/prohibited rule, the other extremity of the line segment of the starting node will always be included in the cycle, if a cycle is found. The line segment containing the starting node is referenced as the *starting edge*.

To prevent finding the same cycle multiple times, a list of remaining candidate edges is kept. Every time a cycle is found, their edges are removed from this list.

3. Enhanced Cycle Search

The computational cost to find all the cycles in a graph is high. To limit the search to cycles likely to correspond to building perimeters (those of smaller total lengths), we modified the base algorithm:

(i) The number of vertices in a path having alternative directions (three or more edges) is limited by a maximum.

(ii) A move from a node v_i is restricted to be along edge (v_i, v_j), if $g_{ij}=1$ in the connection matrix and v_j is not yet discarded by the visited/prohibited rule. Particularly, the first move away from the root node will always traverse an existing line segment.

(iii) At every node, the jumping options (edges in Γ) will be sorted according to the increasing Euclidean distances from the other endpoints of the associate line segments to the starting node. The first branch to be depth-searched will be the one containing the node that minimizes that distance and so on.

4. Elimination of Spurious Cycles

To eliminate likely spurious cycles, the algorithm discards all cycles that contain some other cycle, or that the set of constituent line segments of a given cycle is a subset of those of another cycle.

5. Cycle Filtering for Buildings

Independently of scale, some cycles are more likely to represent buildings than others (see Figure 28).

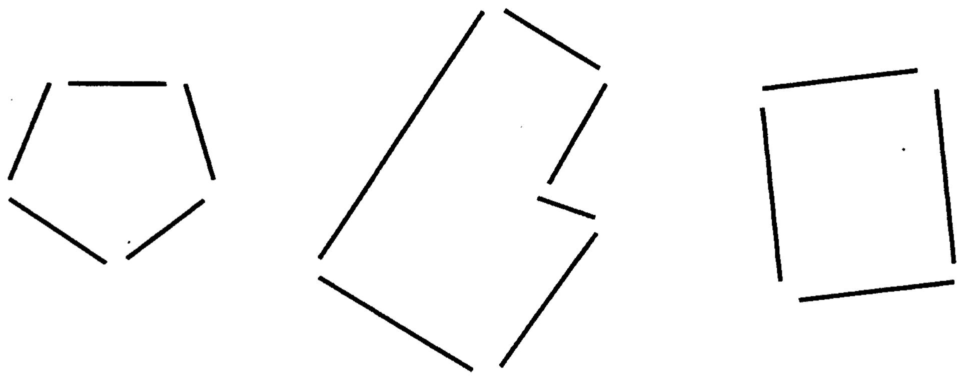


Figure 28: Example of possible detected building candidates from cycles in Γ. Intuitively, the first on the left is not likely to be a building while the two on the right are.

To select buildings among the extracted polygonal paths, some building-likelihood measures are computed from the following assumed properties:

(i) The major part of the polygon perimeter should coincide with (i.e., intersect) original primitive line segments;

(ii) Most of the lateral faces of buildings should be either parallel or perpendicular to largest face of the building; and

(iii) The larger the lateral faces, the less deviation they should exhibit from either the parallel or the perpendicular directions to the largest face of the building.

Deviation of each of these properties from an ideal can be measured. If the building has an unusual shape such as an equilateral triangle or a pentagon, these properties will not hold, but this is not a likely case.

Figure 29 shows an example using the polygon (Q_1, Q_2, ..., Q_M) obtained from the cycle (P_{11}, P_{12}, ..., P_{M1}, P_{M2}).

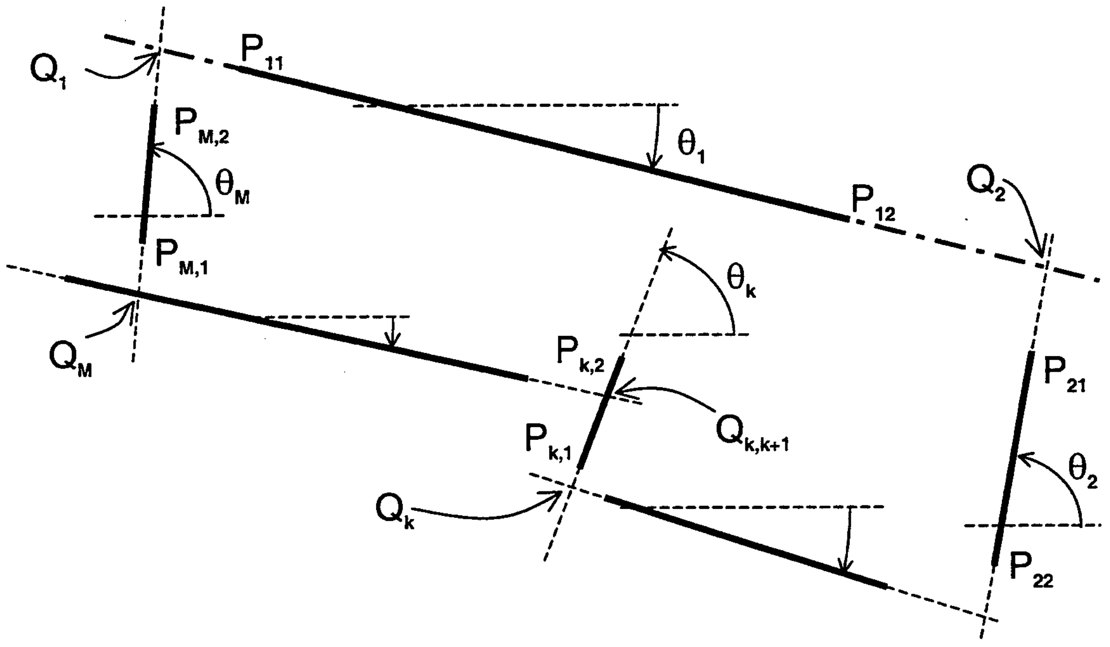


Figure 29: Analysis of polygonal paths formed derived from line segments.

Defining $P_{M+1} \equiv P_1$ and $Q_{M+1} \equiv Q_1$ the three measures are defined as below:

(i) Unsupported perimeter fraction:

$$f = 1 - \frac{\sum_{k=1}^{M} \left| \overline{P_{k1}P_{k2}} \cap \overline{Q_kQ_{k+1}} \right|}{\sum_{k=1}^{M} \left|\overline{Q_kQ_{k+1}}\right|} \tag{19}$$

(ii) Average weighted non-orthogonality factor:

$$\bar{w} = \frac{\sum_{k=1}^{M} \left|\overline{Q_kQ_{k+1}}\right| . \sin(2\left|\theta_k - \theta_{i*}\right| \bmod \pi/2)}{\sum_{k=1}^{M} \left|\overline{Q_kQ_{k+1}}\right|} \tag{20}$$

(iii) Maximum weighted non-orthogonality factor:

$$w_{max} = \frac{\max\{\left|\overline{Q_kQ_{k+1}}\right| . \sin(2\left|\theta_k - \theta_{i*}\right| \bmod \pi/2)}{\max\{\left|\overline{Q_kQ_{k+1}}\right|\}} \tag{21}$$

where i* in equations (20) and (21) is such that $\left|\overline{Q_{i*}Q_{i*+1}}\right| = \max\{\left|\overline{Q_kQ_{k+1}}\right|\}$.

All these measures range in the interval [0, 1]. The closer each is to zero, the more likely that the polygon Q is associated with a building. Higher values of these deviation measures are more acceptable with cycles made of fewer segments. So the detection of buildings will require thresholds Φ_f , $\Phi_{\bar{w}}$ and $\Phi_{w_{max}}$ that are non-increasing functions of the number **n** of line segments in the cycle. Reasonable functions yielding good performance were experimentally determined as shown in Figure 30.

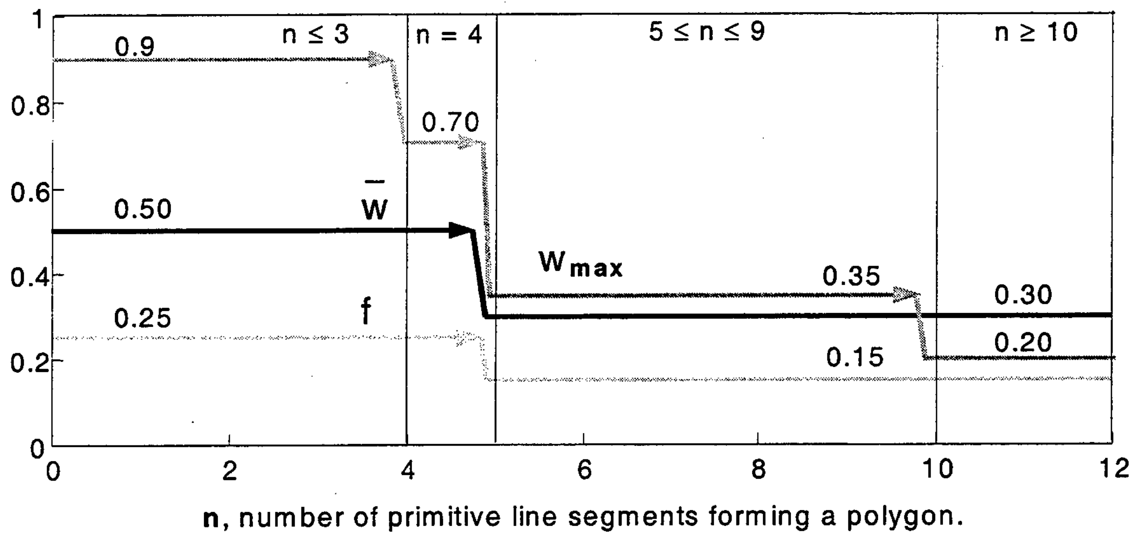


Figure 30: Threshold functions experimentally determined for the computed unsupported perimeter fraction (f), the average weighted non-orthogonality factor ($\bar{w}$), and the maximum weighted non-orthogonality factor (w_{max}).

Figure 31 shows example results of the cycle detection phase. Cycles that satisfy the thresholds for all the three thresholds are filled in black; the others are plotted unfilled. The latter were often found to correspond to non-building man-made structures, like parking lots, open-sky storage areas, or blocks of houses.

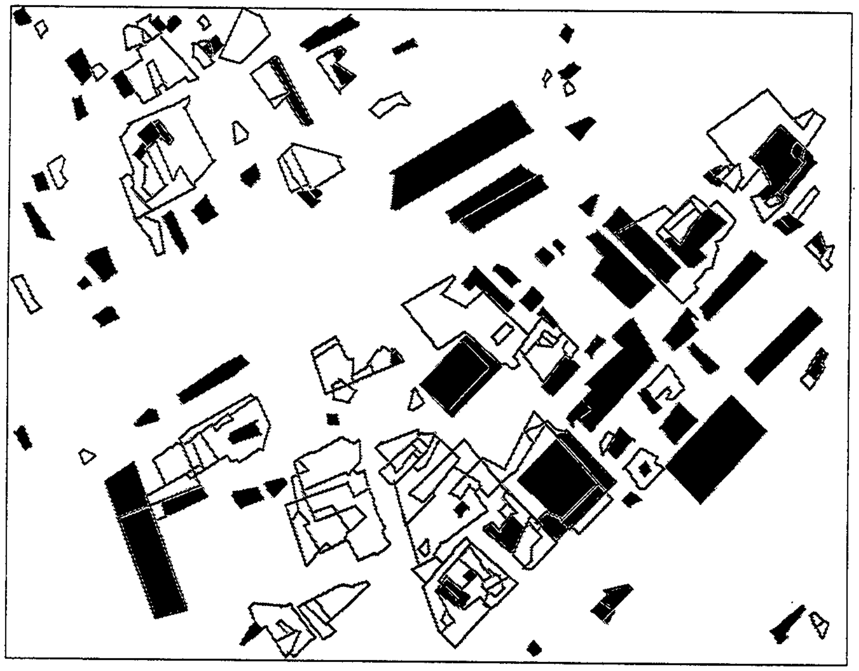


Figure 31: Detected segment cycles in the image of Figure 16; building-like cycles are filled in black.

6. Merging Component Cycles of Buildings

Complex buildings and clusters of buildings will often appear as adjacent and overlapping cycles. So in a final step, all component cycles with non-null intersection (i.e., sharing some line segment) are merged, producing a target table (like Table 2). Entries are the coordinates of the center of mass of each building cluster, its estimated area, average pixel luminance, and standard deviation of the pixel luminance.

```
+-----------------------------------------------------------------------+
|  Building Cluster List for Image in 'H:\MRY\MRY1.tif'                 |
+-----------+---------+---------+------------+---------+--------------+
| Target ID | Coord I | Coord J |  Area (m2) |  Av Lum | Std Dev Lum  |
+-----------+---------+---------+------------------------------------------+
|   00001   |   194.8 |   253.1 |         62 |    33.2 |         23.3 |
|   00002   |   234.9 |   245.3 |        328 |    74.4 |         30.5 |
|   00003   |    78.4 |   235.1 |         78 |   170.0 |         37.3 |
|   00004   |   115.9 |   211.8 |        146 |   132.2 |         41.1 |
|   .....   |   ..... |   ..... |      ..... |   ..... |        ..... |
|   00075   |    99.3 |   216.8 |         89 |    98.7 |         37.5 |
+-----------+---------+---------+------------------------------------------+
```

Table 2: Example target table produced by the program, listing the probable building clusters found in Figure 4 and properties.

In this simplified approach, shadow detection is not implemented, and shadows may be included with building clusters. This is a problem only if the sun angle is low relative to the building height/length ratio and the shadow is aligned with a face of the building. Otherwise, the cycle containing that shadow will have its non-orthogonality factor increased, which will tend to cause its rejection in the cycle filtering phase (explained in IV.F.5).

In Figure 32, recognized building clusters are shown with homogeneous random color. A white cross is placed at the center of each cluster. A total of 75 clusters were formed from 97 detected cycles.

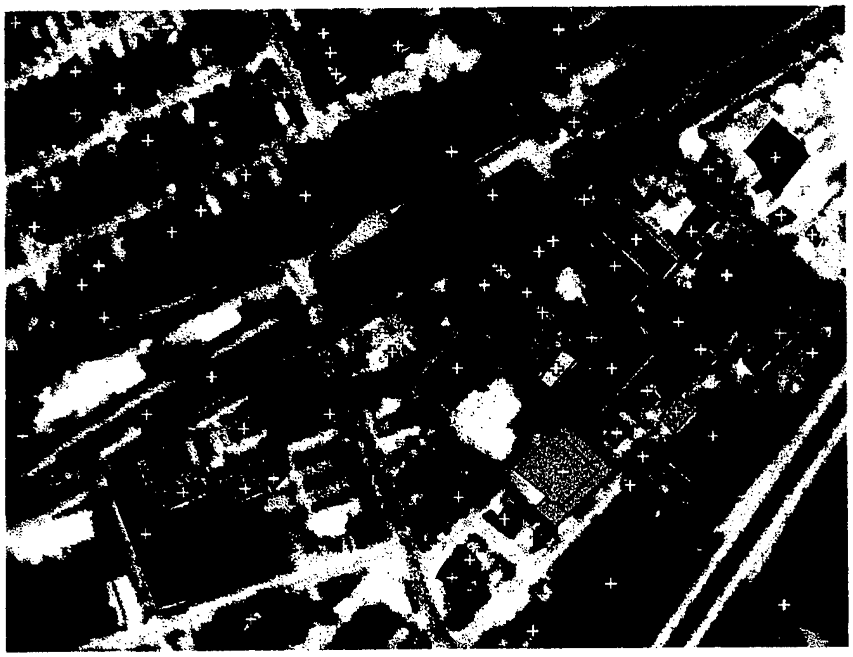


Figure 32: Plot of recognized probable building clusters, after merging of component cycles. Random colors are assigned to clusters for improved visualization.

G. USING NEURAL NETWORKS

1. False Alarm Reduction with Neural Networks

The Φ_f, $\Phi_{\bar{w}}$ and $\Phi_{w_{max}}$ thresholds functions (see Figure 30) that help recognize a building cycle were chosen heuristically. If enough training images are available, we could improve building cluster recognition by training a feed-forward artificial neural network (Multi-Layer Perceptron) for the task [Ref. 19], as shown in Figure 33. The network could take as inputs at least the quantities f, $\bar{w}$, w_{max} defined in equations (19) through (21), and n, the number of line segments, yielding binary outputs c_i to distinguish classes of similarly orthogonal-shaped man-made objects.

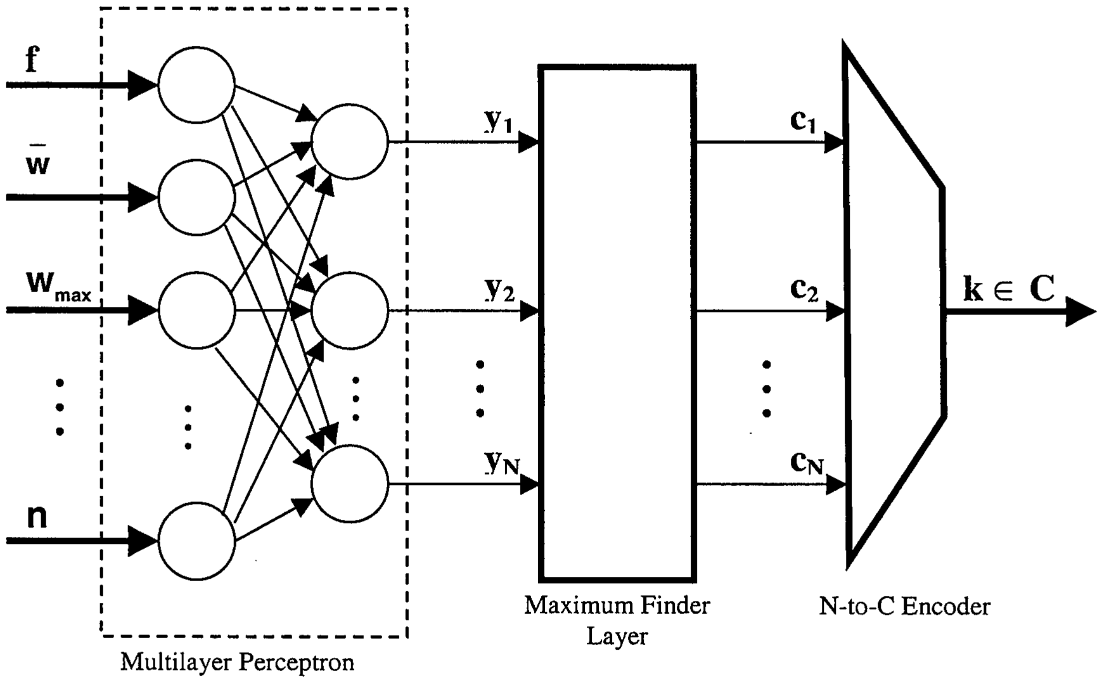


Figure 33: Classifier architecture using neural networks for the classification of cycles in the endpoint graph.

When edges along the contour of an object are so degraded that no cycle around it is found, this architecture would not be helpful. So the neural network would only significantly reduce the false alarm ratio, not the miss ratio.

2. Selecting More Parameters for the Classification of Cycles

Other features associated with the region enclosed by the detected polygon could be used as inputs for the neural network to improve its performance:

(i) Size-related features, for example: area of the region, perimeter, moment of inertia radius, maximum distance from center of inertia, and circumvention radius.

(ii) Luminance-related features: total brightness, average brightness, estimated standard deviation of the brightness, displacement of brightness center (mass center weighted by luminance) from mass center, and moment of inertia radius using the brightness as the density function.

(iii) Other shape related features: the ratio of the maximum and minimum coefficients of the Radon Transform of the object, the ratio of the moment of inertia radius to the circumvention radius, the ratio of the displacement of center of brightness to circumvention radius ratio, and the sphericity, defined as four times the area divided by the square of the perimeter.

Preliminary tests [Ref. 20] showed promising results with neural networks for feature analysis and classification of objects in aerial photographs. In this study, buildings, road sections and trees were correctly differentiated in essentially all cases tested (10 buildings, 10 paved road sections, 10 unpaved roads, and 10 trees), with only a slight confusion occurring in between paved and unpaved roads. For meaningful results however, much larger training and validation data sets need to be used.

A neural network could also recognize shadows of objects. This would improve the accuracy in the estimation of the center of mass of each building cluster, and could also facilitate the three-dimensional modeling of the scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS FROM EXPERIMENTATION

A. QUALITY ASSESSMENT OF THE MODEL

The quality of the recognition of building clusters can be evaluated by two error measures: The false alarm ratio (false positive recognition) and the miss ratio (false negative recognition). The first is obtained by dividing the number of incorrectly labeled building clusters by the total number labeled; the second, by dividing the total number of building clusters not recognized by the total number of building clusters. Both numbers should be zero for a perfect recognition. Because the resolution of the NAPP images is only slightly better than the minimum necessary for recognition of buildings (see section II.F.1), houses and small buildings were not counted (when missed) in evaluating our automated analysis.

Appendix A contains the detailed description of the primary test image used (Figure 4). This image includes residential areas as well as public, commercial and industrial buildings. Two small regions had to be excluded from the results due to unavailability of reference data. The reasons in these two cases were:

(i) Part of a block was completely remodeled since the photograph was taken six years ago, and we could not recover the original layout of that area; and

(ii) One small area (south of US Highway 1), of difficult access for a walk-through visit, was not inspected.

The recognition statistics for building clusters are summarized in Table 3. The false positive (e_p) and false negative (e_n) recognition ratios were then 0.133 and 0.177 respectively, similar to those obtained by others using different edge-based techniques [Ref. 21, 22].

Number of building clusters **correctly** recognized	$n_C = 65$
Number of building clusters **incorrectly** recognized	$n_W = 10$
Number of building clusters **missed** (i.e., not recognized)	$n_M = 14$

Table 3: Summary of performance of our building recognition technique.

1. False Negative Errors

False negative errors (missing buildings) can be one of the following:

(i) errors due to deficient primitive line-segment extraction;

(ii) errors due to splitting of line segments of a building at cluster boundaries; and

(iii) errors due to oversimplification in the cycle-filtering criterion.

Errors of type (i) were due to the limitations of the edge extraction algorithm (Canny's). When edges are interrupted along the sides of buildings in consequence of lack of contrast or obstacles, the line segments extracted will be fragmented, and the building may be missed. And with too-small structures, the corners tend to be rounded, interfering with the line-segment extraction. An example is buildings i and j in Figure 34. This error was the cause for 50% (7/14) of the missing buildings (a, c, e, h, i, j and l), and could be reduced by improving the edge-finding algorithm.

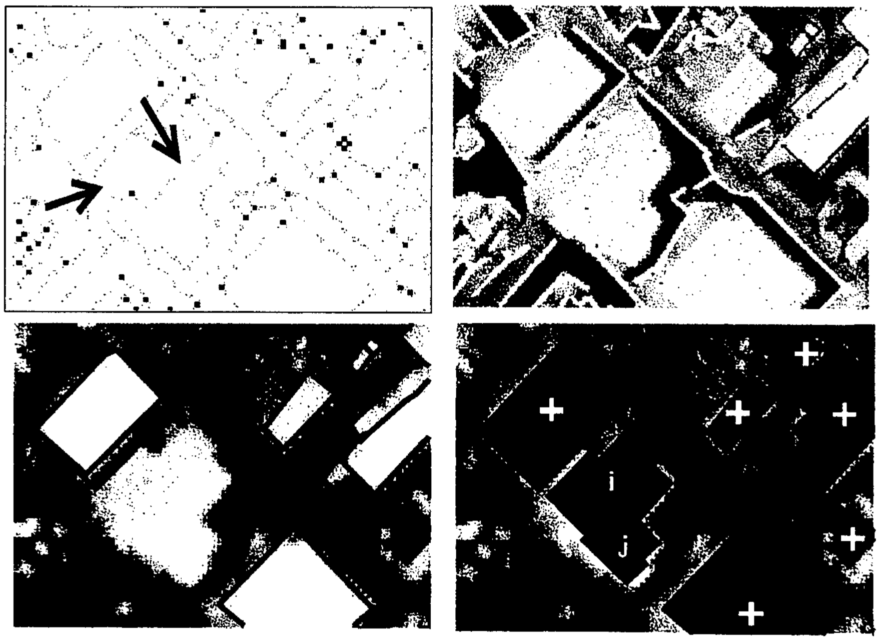


Figure 34: Detail showing examples of missing buildings (i and j) due to defective line edge and line segment extraction.

Errors of type (ii) are due to imperfect line-segment clustering preceding connectivity analysis of the endpoints graph. Figure 35 shows a sequence of steps causing failure to recognize buildings **b** and **d**. Since cycles are only searched within each cluster of line segments, problems occur when a building contour is split into different clusters. These errors, responsible for 14% (2/14) of the missing buildings in the tested image, could have been eliminated if the line segment clustering were suppressed, but then the complexity of the algorithm would increase considerably.

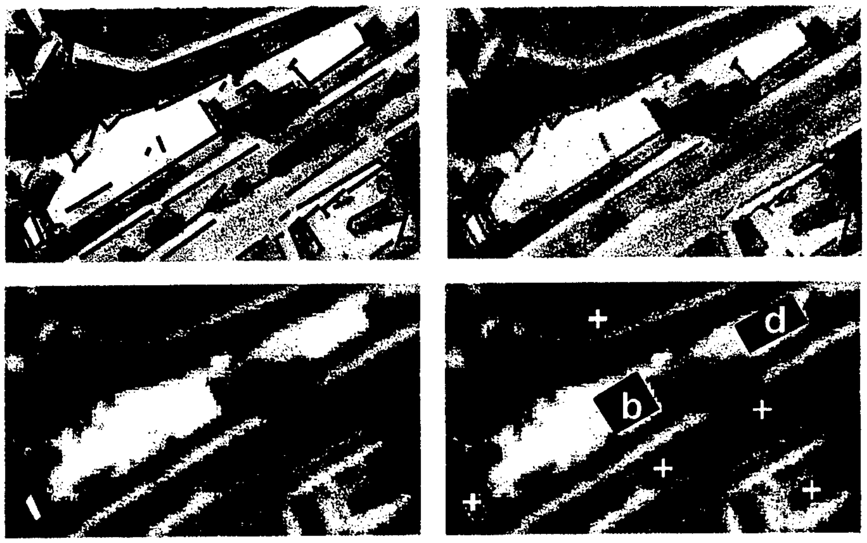


Figure 35: Examples of missing buildings (b and d) due to the splitting of perimeter line segments at cluster boundaries.

Finally, errors of type (iii) which account for 35% (5 /14) of the misses and a 6% of the overall false negative recognition ratio, seem to be intrinsic to the devised cycle filtering algorithm. To eliminate them a major modification in the algorithm is necessary.

2. False Positive Errors

All the false positive errors were entities with contours of plausible building shapes: Three wooded road divider sections (targets id 50, 51 and 52), two parking lots (targets id 30 and 64), one partially fenced private drive (target id 66), one tree surrounded by a paved path on the corner of a block (target id 1), one school playground (targets id 37), and two square bare ground areas (target id 65 and 71). All of these except the last two were man-made structures, which makes the errors less serious. These errors were to be expected, for the algorithm used does not consider luminance, texture, or relationships to others neighbor structures, all of which could improve performance.

B. OTHER ERROR MEASURES

Beyond the correctness of the building recognition, the correctness of position determination also matters. In our experiment the field determination did not include the precise position and area of the buildings in the testing data set. However, we did confirm that all estimated centers of targets were within the respective actual building clusters.

C. IMPLEMENTATION ISSUES

All the programs were implemented in Matlab version 5.3, running under the Windows NT 4.0 operating system, a language that offers an interpreter command interface and debugging facilities. All the images were stored in the uncompressed gray-level Tagged Image File Format (TIFF), a widely used bitmap file format.

The edge-based approach for finding buildings is computationally expensive. For instance, roughly 2 million operations are required to find a single 51 x 51 pixel square pattern in a simple test image of 100 x 100 pixels (see Figure 36).

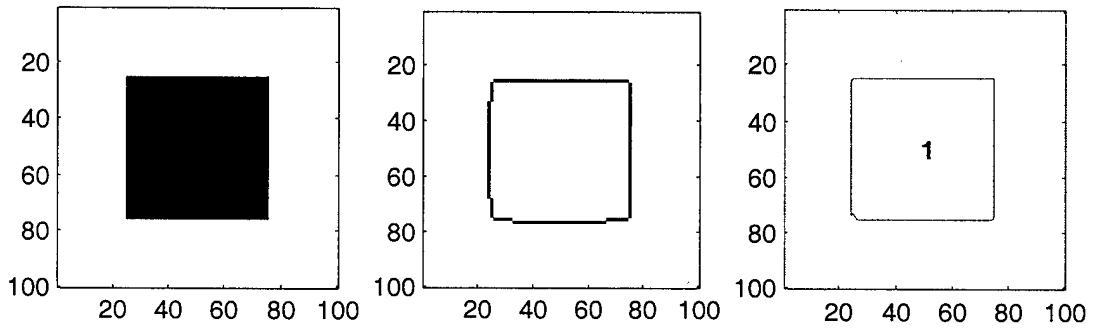


Figure 36: Testing the algorithm on a simple artificial image: test pattern, extracted edges, and recognized shape.

For the main testing image used (Figure 4), 340 x 260 pixels in size and approximately 3 square meters per pixel, the number of necessary operations to complete the algorithm was of 7.4 x 10^8, including the graphic output. Running Matlab on a Pentium II processor at 266 MHz with 64MBytes of memory (about 72,000 sustained floating-point operations per second), this computation took 2 hours and 50 minutes. If a compiled version of the system could be implemented and run on a 1 Mflops platform, a modest computing performance for today standards, the system should process around 380 square meters of urban area per second. We have focussed however on the algorithms, not optimization of the code; much could be done to improve the efficiency and the speed of the programs.

In total, about 4000 lines of source code were created to implement the algorithm, including comments and not including the source code of the powerful Matlab libraries used. Program listing is included in Appendix B.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

An algorithm for finding building clusters in orthorectified aerial photographs was implemented and tested on an urban area. The technique used was based on the connectivity analysis of a graph derived from the geometric relationships among endpoints of line segments that model edges; the segments were extracted from the image with the Radon Transform. The connectivity analysis reveals cycles in the graphs that, once filtered for spurious closed paths, indicate candidate buildings.

The 2-meter panchromatic resolution of the test image is barely enough for a human expert to recognize the smaller buildings. Under these conditions, the obtained false alarm and miss ratios were respectively 13% and 18%, not counting errors on houses and very small buildings. Although not perfect, these figures should be able to reduce substantially the workload of human analyst in a computer-aided environment.

Automatic aerial image analysis is a complex problem. The complete validation of the algorithms and ideas proposed here requires more testing sets and extended conditions. In our tests, the dominant cause for false negative errors (misses) was the edge detection algorithm (Canny's), while the false positive errors where due to the assumption that shape alone can determine buildings. We believe that our edge-based recognition of man-made structures will produce better results if combined with region-based techniques. One way to correct this is the consideration of luminance information, perhaps by using a feed-forward neural network for postprocessing.

While the speed of our algorithm may be disappointing, due to the number of mathematical operations involved, there is much parallelism opportunity do be exploited that could make it much faster. It also could be helpful the introduction of a line-segment pruning step before the clustering of primitive line segments, to decrease the contour density, similarly as done by Paparoditis et Al. [Ref. 23].

With the increasing availability of high-resolution imagery from both military and commercial sensors, better resolutions that 2-meter will soon be abundantly available. Since we can recognize buildings even at coarser resolution with our approach, it should be able to recognize additional details within buildings and other man-made structures when using higher-resolution images.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. EXPERIMENT NOTEBOOK

Figure 37 shows the building clusters detected in the test image (Figure 4). Statistics on them are shown in Table 4.

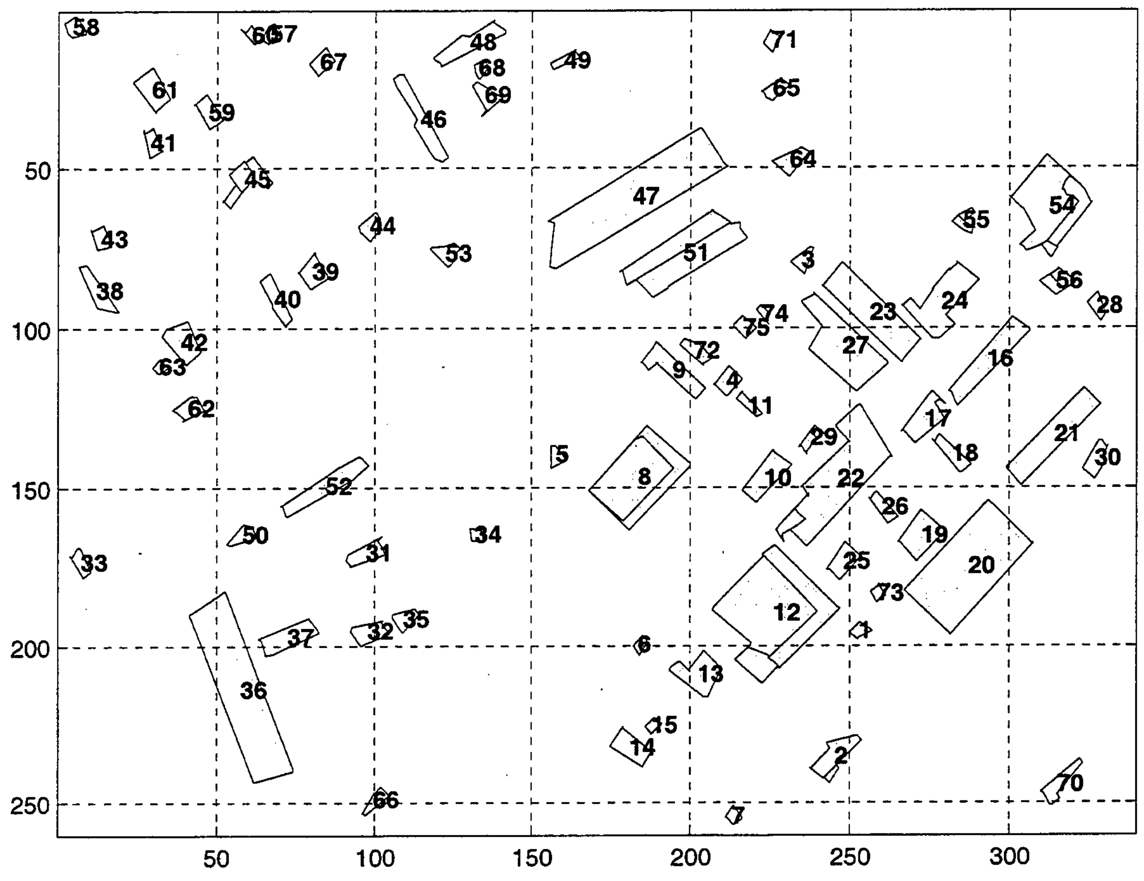


Figure 37: The 75 building clusters found in Figure 32 by our program.

```
+-------------------------------------------------------------------------+
|   Building Cluster List for Image in 'H:\MRY\MRY1.tif'                  |
+-----------+---------+---------+-------------+---------+-----------------+
| Target ID | Coord I | Coord J |  Area (m2)  | Av Lum  |   Std Dev Lum   |
+-----------+---------+---------+-------------+---------+-----------------+
|   00001   |   194.8 |   253.1 |          62 |    33.2 |            23.3 |
|   00002   |   234.9 |   245.3 |         328 |    74.4 |            30.5 |
|   00003   |    78.4 |   235.1 |          78 |   170.0 |            37.3 |
|   00004   |   115.9 |   211.8 |         146 |   132.2 |            41.1 |
|   00005   |   139.5 |   157.3 |          57 |   235.6 |            17.7 |
|   00006   |   199.3 |   183.8 |          46 |    99.9 |            28.7 |
|   00007   |   253.9 |   213.6 |          51 |    65.5 |            26.1 |
+-----------+---------+---------+-------------------------------------------+
```

Target ID	Coord I	Coord J	Area (m2)	Av Lum	Std Dev Lum
00008	146.3	183.5	1689	187.2	80.0
00009	112.7	194.7	402	154.9	29.7
00010	146.7	223.5	352	174.6	25.2
00011	124.0	218.2	51	146.5	43.4
00012	189.2	226.3	2766	178.5	91.5
00013	208.6	202.4	382	78.1	45.1
00014	232.1	180.9	276	129.1	52.2
00015	225.0	188.0	48	103.8	39.3
00016	109.4	292.9	607	143.7	29.2
00017	128.2	273.2	333	73.8	23.0
00018	139.3	282.1	177	203.1	58.9
00019	164.7	272.3	391	235.7	38.0
00020	174.7	287.0	2400	182.8	40.7
00021	133.0	314.2	838	202.8	43.4
00022	146.8	246.1	1884	194.8	62.6
00023	94.5	256.3	946	243.1	31.4
00024	91.3	278.6	794	231.2	32.5
00025	173.0	248.0	192	63.0	24.0
00026	155.9	260.2	108	188.7	56.9
00027	105.3	247.7	1114	220.3	52.3
00028	92.9	327.4	105	226.4	23.0
00029	134.2	237.9	81	134.6	44.0
00030	140.9	327.2	155	70.3	24.2
00031	170.3	97.1	174	66.2	21.7
00032	195.1	97.7	189	226.6	46.2
00033	173.0	7.5	116	203.8	53.1
00034	164.6	131.8	48	124.3	38.0
00035	191.1	109.1	109	227.9	33.8
00036	213.5	57.5	2622	223.1	40.1
00037	196.6	72.4	350	221.6	37.4
00038	87.7	12.4	206	154.3	32.9
00039	81.7	79.9	195	95.0	37.6
00040	90.3	68.2	220	66.4	28.2
00041	41.5	29.3	89	106.0	40.1
00042	103.7	38.6	298	101.0	48.4
00043	71.5	13.8	108	174.0	37.8
00044	67.5	98.2	131	44.1	37.1
00045	52.9	58.7	333	91.7	31.7
00046	34.2	114.0	361	70.3	36.5
00047	58.5	181.6	2703	206.3	49.1
00048	10.2	129.8	355	126.4	38.9
00049	15.7	160.0	87	131.6	49.9
00050	164.3	57.9	108	72.0	23.2
00051	76.2	198.0	1442	47.9	29.3
00052	149.1	84.2	451	30.4	19.0
00053	76.0	122.3	130	91.8	29.2
00054	61.6	312.5	1328	229.5	44.0
00055	66.3	285.4	98	105.6	15.5
00056	85.0	314.8	152	140.6	47.5
00057	7.2	66.7	70	93.0	33.6
00058	4.9	5.0	95	79.8	32.1

```
+-----------+---------+---------+------------+---------+--------------+
| Target ID | Coord I | Coord J | Area (m2)  | Av Lum  | Std Dev Lum  |
+-----------+---------+---------+-------------------------------------+
|   00059   |    31.7 |    47.1 |        150 |   135.5 |         35.2 |
|   00060   |     7.8 |    60.8 |         74 |    92.9 |         30.3 |
|   00061   |    24.8 |    29.4 |        299 |    72.0 |         44.3 |
|   00062   |   124.7 |    40.8 |        122 |    59.5 |         14.9 |
|   00063   |   111.3 |    31.9 |         38 |    77.6 |         20.6 |
|   00064   |    47.0 |   231.2 |        154 |   192.8 |         38.0 |
|   00065   |    24.9 |   226.1 |         84 |   107.1 |         43.5 |
|   00066   |   248.5 |    99.6 |         82 |    45.5 |         17.8 |
|   00067   |    16.3 |    82.5 |        117 |   109.1 |         34.1 |
|   00068   |    18.2 |   132.6 |         46 |    46.6 |         24.2 |
|   00069   |    26.6 |   134.8 |        138 |    56.8 |         18.3 |
|   00070   |   244.1 |   315.2 |        198 |    71.6 |         33.0 |
|   00071   |     9.5 |   225.2 |         57 |    92.1 |         36.2 |
|   00072   |   106.7 |   201.3 |        120 |    78.5 |         32.1 |
|   00073   |   182.9 |   258.6 |         52 |    50.3 |         22.2 |
|   00074   |    95.0 |   222.6 |         36 |    41.3 |         24.7 |
|   00075   |    99.3 |   216.8 |         89 |    98.7 |         37.5 |
+-----------+---------+---------+-------------------------------------+
```

Table 4: Target table automatically produced by program. Target ID refers to labels in Figure 37. I and J coordinates define the center of mass of the cluster, other entries are the area in squared meters, the average pixel luminance, and the standard deviation of the pixel luminance.

After a field survey, the following facts were established (see Figure 38):

(i) Of the 75 targets found, 10 did not correspond to any kind of building, building cluster, or housing area (false positives). These are given in Table 5:

Target ID	Description
1	Tree surrounded by squared path on the corner of Encina Ave.
30	U-Haul parking lot, full of trucks parked in parallel at time of survey.
37	Del Monte Elementary School playground (rectangular, bare ground).
50	Section of road division lot with trees (boulevard) at Del Monte Ave.
51	Section of road division lot with trees (boulevard) at Del Monte Ave.
52	Section of road division lot with trees (boulevard) at Del Monte Ave.
64	Parking lot with bare ground terrain.
65	Square shaped bare ground terrain partially surrounded by fence.
66	Private drive delimited by fence.
71	Square shaped bare ground terrain.

Table 5: False positive building detection.

(ii) Among the remaining 65 patterns correctly identified, those which are not housing areas are listed in Table 6:

Target ID	Description
3	Annex of Willie's & Fraley Auto Repair (2232 Del Monte Ave.)
4	Dairy Producers Office
5	Advantage Auto Repair & Muffler
8	Tileco Ceramic (2110B Del Monte Ave.)
9	Greg Bean Auto Servicing (2200 Del Monte Ave.)
10	Ewing Irrigation Products
12	Store USA main building
16	McCuhe Audio-Visual
17	Allied Storage Warehouse
18	Wilson's Plumbing And Heating
19	C & C Repair
20	Miller Moving & Storage Co. (on Dela Vina Ave.)
21	Miller Moving & Storage Co. (on Ramona Ave.)
22	Allied Van Lines
23	Willie's / Fraley Auto Repair
24	Redwood Heating
26	Hubbard Plumbing
27	Moving & Storage Wermuth & Cahoon
28	Foreign Affairs office
29	Old garage for Allied (now demolished, but present at time of photo)
31	Aquarius Dive Shop
32	Del Monte Elementary School Building
33	Monterey Ironworks Annex
35	Del Monte Elementary School Building
36	Del Monte Glass
47	Maris (2101 Del Monte Ave.)

56	United Rentals
72	Dairy Producers Office
75	Dairy Producers Office

Table 6: Commercial and industrial buildings correctly detected.

(iii) Some major non-residential buildings were also missed (false negatives). They are listed in Table 7 below:

Target ID	Description
a	Skate Arena
b	Monterey Ironworks main building
c	Linda Motel
d	Natale's Auto Service
e	Del Monte Elementary School Building
f	Del Monte Elementary School Building
g	Del Monte Elementary School Building
h	Del Monte Elementary School Building
i	Monterey Gymnastics (220 Dela Vina Ave.)
j	Storage USA Annex
k	Dairy Producers wooden roof open storage
l	Conte's Auto Repair
m	ABC Glass
n	City Community Chapel

Table 7: False positive building detection.

(iv) The dashed area on the northeast block delimited by Del Monte Ave. and Ramona Ave. suffered recent remodeling; old constructions were demolished and newer buildings were erected since the year of the photo (1993). So all events inside that area were ignored. An area at south of US 1 was also ignored because of the difficulty of access.

In Figure 38, building clusters are plotted with a color schema for easy visualization of the results, in a similar way to that used in [Ref. 24]. Those identified with letters and painted in blue are buildings missed. Those in red are those erroneously recognized (false alarms). Areas in green are correctly recognized buildings, building clusters or other housing.

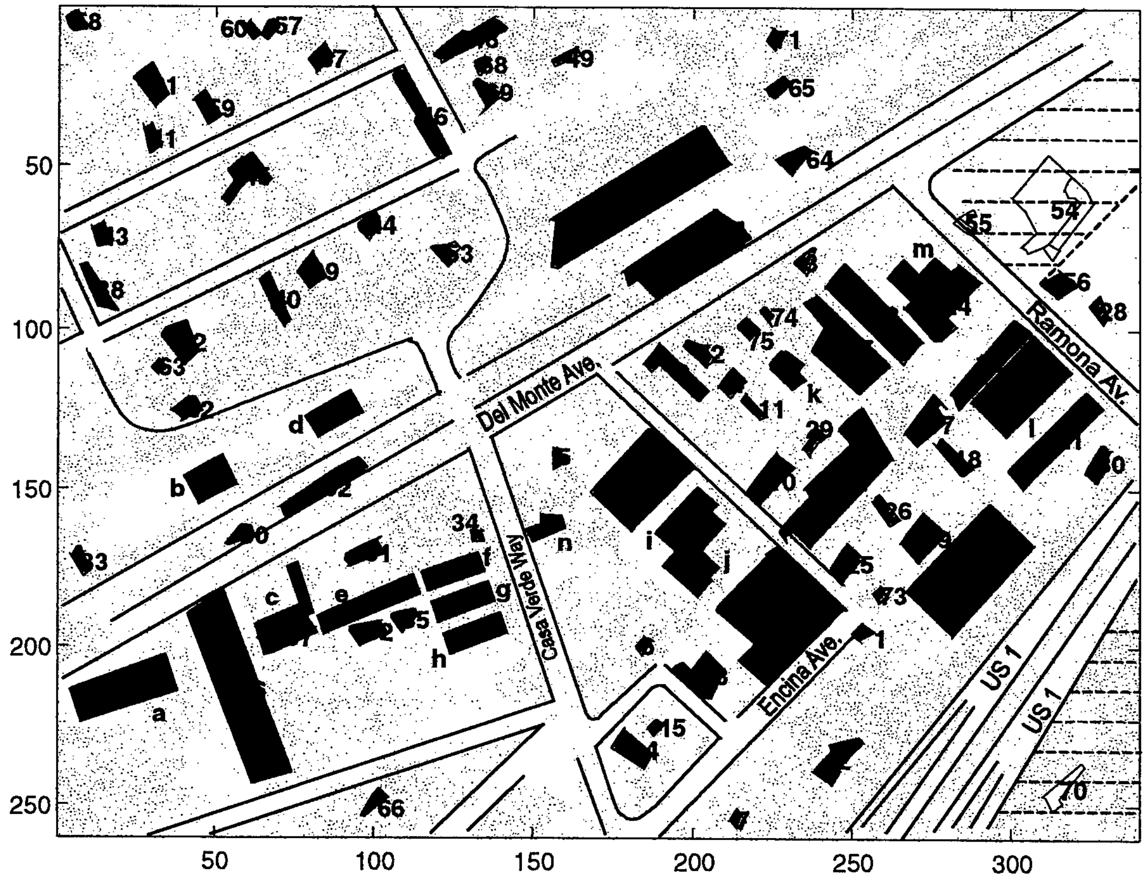


Figure 38: Reference information for the scene.

Summarizing, from the 75 patterns recognized as buildings or housing structures, only 65 were actually buildings, while 14 other relevant buildings (non-residential) were missed. At the marginal resolution of the image, false negative recognition of small buildings such as residential houses should not be penalized. This gives a rough estimate of false positive recognition ratio of (75 – 65) / 75 ≅ 13% and a false negative recognition ratio of 14 / (65 + 14) ≅ 18 %.

APPENDIX B. PROGRAM LISTING

```
function [BuildingCluster, PLCluster, totalElapsedTime, numOps] = ...
   find_building_clusters(FilePath, FileName, pixelLength,...
   logResultsFlag)
%
%  Usage:
%
%  [BuildingCluster, PLCluster, totalElapsedTime, numOps] = ...
%     find_building_clusters(FilePath, FileName, pixelLength, logResultsFlag)
%
%
%  Description:
%
%  Finds probable building clusters in an orthorectified aerial
%  image given by the gray scale TIFF file called 'FileName'.
%

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'find_building_clusters.m'
% (Main function call for the building finding routine)

Flops0 = flops;

Time0 = clock;

%================================================================%

global CornerLookUpTable BifurcLookUpTable FillStraightGapsLookUpTable
createCornerLookUp

global logResults
logResults = logResultsFlag;

global BTotal

% limit value of differencial angle to be considered paralel/orthogonal:
global cosParalelLimit cosParalelLimitForEquiv limitDist
global cosColinearLimit1 cosColinearLimit2 tanLimitForMerging SINMAX1 SINMAX2

global WorkingDirectory

% initialization of constants
```

```
cosParalelLimit = cos(5.5*pi/180);          % +/-5.5 degrees
cosParalelLimitForEquiv = cos(15*pi/180); % +/-10 degrees
tanLimitForMerging = tan(20*pi/180);
limitDist = 2;

% define 1 to plot contours, 0 not to plot.
showContours=1;

% set working directory
WorkingDirectory = FilePath;

% read image TIFF file
A = imread([FilePath FileName],'tif');

PhaseI    % Edge extraction from image:  B{} <--- edge(A)

PhaseII   % Extract primitive lines from e dges: PL <--- primitives(B{})
   %  load([FileName '.PhaseII.mat']);

PhaseIIA  % Cluster primitive lines:  PLCluster <--- PL
   % load([FileName '.PhaseIIA.mat']);

for PLClusterNumber=1:length(PLCluster),
   PL=PLCluster{PLClusterNumber};

   PhaseIII % Enhance primitive lines: PL <--- enhanced PL

   PLCluster{PLClusterNumber} = MergedPL;
end

figure(8)

plotwithlines(A,PLCluster,mod([1:size(Clusterization,2)],2)+1,colors);

numBuildingsInCluster = zeros(1,length(PLCluster));
disp('------------------------------------------------------------------------
');
disp(['Begining of building search phase for image in: ''' FileName '''']);

for PLClusterNumber=length(PLCluster):-1:1,
   PL=PLCluster{PLClusterNumber};

   % PhaseIVA: Find cycles in PL graph
   [PL, loop, PLinLoop, DecomposedPLLoops, Indexes,...
      contourLoop, supportedFraction, shaperErr, errMax, IJCoordinates] = ...
          PhaseIVA(A, PL, logResults, FileName, PLClusterNumber);

   PLCluster{PLClusterNumber}=PL;

   numBuildingsInCluster(PLClusterNumber)=length(contourLoop);

   if numBuildingsInCluster(PLClusterNumber) > 0
      title(['Cluster ' int2str(PLClusterNumber) ' of '
int2str(length(PLCluster))...
         ': ' int2str(length(contourLoop)) ' Building candidates found.'])
       % pause(1)
   end

   for bNum=1:length(contourLoop),
      CandidatePolygon{PLClusterNumber,bNum} = contourLoop{bNum};
      shapeError{PLClusterNumber,bNum} = shaperErr{bNum};
      shapeMaxError{PLClusterNumber,bNum} = errMax{bNum};
      sFrac{PLClusterNumber,bNum} = supportedFraction{bNum};
      PLinPolygon{PLClusterNumber,bNum} = PLinLoop{Indexes{bNum}};
      cycleSummary{PLClusterNumber,bNum}=loop{Indexes{bNum}};
   end
end

debugMode=0;
PlotWithImageInBackground=0;
PlotContourOnly=1;

[Building, figHandle] = ...
   selectbuildingcandidates(A, PlotWithImageInBackground, PlotContourOnly,...
      CandidatePolygon, PLinPolygon, cycleSummary, shapeError,
shapeMaxError,...
         sFrac, numBuildingsInCluster, size(A), debugMode);

pause(1)
Building

if logResults
   hgsave(gcf, [FileName '.Buildings.Fig']);
   save([FileName '.PhaseIVA.mat'])
   % print
end

% load([FileName '.PhaseIVA.mat']);

figure(8)

% Analyze building clusters
imageBackground = 1;
numberPlot = 1;

[NumberOfBuildingClusters, BuildingCluster] = ...
   PhaseVA(A, imageBackground, Building, PLCluster,...
   pixelLength, FileName, 0, numberPlot)

%==================================================================%

% measure performance

Flops1 = flops;

Time1 = clock;

totalElapsedTime = etime(Time1, Time0);
numOps = Flops1 - Flops0;

disp(['Total elapsed time = ' num2str(totalElapsedTime) 's'])
disp([ '(' num2str(numOps) ' float point ops @ ' ...
   num2str((numOps)/(totalElapsedTime*1000)) ' kflops )' ] )

%==================================================================%
% End of file 'find_building_clusters.m'
```

function PL = bestline(r, rCenter);

```
%
% function PL = bestline(r, Center);
%
% PL = [theta, d, base, LimitI, LimitJ]'
%
% Description: Computes the best line passing through the on-pixels
%              in binary image r, minimizing the sum of the squared
%              distances from pixels to the line. 'Center' is the
%              point used as a reference for computing 'd' and the
%              base point 'base'. 'ELRatio' is a measure of the
%              quality of the adjustment.

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'bestline.m'

%----------------------------------------------------------------%

% find the non zero pixels:
[ISet, JSet] = ind2sub(size(r),find(r>0));
P=[ISet JSet];

% compute n number of non zero pixels
n=size(P,1);

% Avoid integer center coordinates guarantee A is not 0
Center = rCenter - sqrt(2)/2;

% find distance of each pixel to Center
Disp=P-ones(n,1)*Center;
D=sqrt(sum((Disp.*Disp)'))';
% find the sin & cos of each pixel
SinP=Disp(:,2)./D;
CosP=Disp(:,1)./D;

ThetaP=atan2(SinP,CosP);
% Angle in degrees = 180*ThetaP/pi

% consider the oriented distance to each pixel
D=-D.*sign(Disp(:,1).*CosP + Disp(:,2).*SinP);

% repeat for rCenter
% find distance of each pixel to Center
ActualDisp=P-ones(n,1)*rCenter;
ActualD=sqrt(sum((ActualDisp.*ActualDisp)'))';
% find the sin & cos of each pixel
```

```
ActualSinP=ActualDisp(:,2)./D;
ActualCosP=ActualDisp(:,1)./D;
ActualThetaP=atan2(ActualSinP,ActualCosP);
% consider the oriented distance to each pixel
ActualD=-ActualD.*sign(ActualDisp(:,1).*ActualCosP +
ActualDisp(:,2).*ActualSinP);

% compute the best tangent of the best angle
% by solving a degree two equation A.t2 + B.t + C = 0
DCos=D.*CosP;
DSin=D.*SinP;
nDSinCos=n*sum(DCos.*SinP);
A=nDSinCos-sum(sum(DSin*CosP'));
B=n*sum(D.*(CosP.*CosP - SinP.*SinP)) - sum(sum(DCos*CosP' - DSin*SinP'));
C=sum(sum(DCos*SinP'))-nDSinCos;
if A<0
   A=-A;
   B=-B;
   C=-C;
end

SDelta=sqrt(B*B-4*A*C);
theta=[atan2(-B-SDelta,2*A) atan2(-B+SDelta,2*A)]';
% 180*theta/pi

d=zeros(2,1);
for k=1:2,
   d(k)=sum(ActualD.*cos(theta(k) - ActualThetaP))/n;
end

% two hypotesis are to be tested:
% (theta=theta(1) and d=d(1))  or (theta=theta(2) and d=d(2))
error=zeros(2,1);
e=zeros(n,2);
e(:,1)=ActualD.*cos(theta(1)-ActualThetaP) - d(1);
error(1)=e(:,1)'*e(:,1);
e(:,2)=ActualD.*cos(theta(2)-ActualThetaP) - d(2);
error(2)=e(:,2)'*e(:,2);

[eSort, eIndex]=sort(error);
kMin=eIndex(1);
theta=theta(kMin);
d=d(kMin);

uTheta = [-sin(theta) cos(theta)];
base   = rCenter - d*[cos(theta) sin(theta)];

% compute de projection of the points on the best line
Y=-ActualDisp(:,1);
X=ActualDisp(:,2);
YSinXCos=Y*sin(theta) + X*cos(theta);
XProj=cos(theta)*YSinXCos - d*sin(theta);
YProj=sin(theta)*YSinXCos + d*cos(theta);
LimitX=[min(XProj) max(XProj)];
LimitY=[min(YProj) max(YProj)];
if theta<0
   LimitY=fliplr(LimitY);
end
LimitI=rCenter(1)-LimitY;
LimitJ=rCenter(2)+LimitX;
```

```
iProj=rCenter(1)-YProj;
jProj=rCenter(2)+XProj;

PL = [theta d base LimitI LimitJ]';

if abs(theta)<pi/4
   pattern=sign(e(:,kMin));
else
   [ISetSorted, IndexSortingI]=sort(ISet);
   pattern=sign(e(IndexSortingI,kMin));
end

E=max(abs(e(:,kMin)));
ELRatio=E/lengthOfPL(PL);

% if debugging, uncomment the line below:
% plotAdjustment(r,E,ELRatio,LimitI,LimitJ,ISet,JSet, rCenter, iProj,jProj);

function plotAdjustment(r, E, ELRatio, LimitI, LimitJ, ISet, JSet,...
   Center, iProj, jProj)
%
% plots the resulting line segment
%
%subplot(122)

clf
% r2 = uint8(3*double(r)+16*(double(auxSeg)-double(r))+double(bigMask));
% imagesc(r2,[0 20])
imagesc(r,[0 2])
colormap(1-gray)
axis image

% axis([min(JSet)-0.5 max(JSet)+0.5 min(ISet)-0.5 max(ISet)+0.5])
% axis([min(JSet+1)-1.5 max(JSet+1)+1.5 min(ISet+1)-1.5 max(ISet+1)+1.5])
hold on
h=line(LimitJ,LimitI);
set(h,'Color',[0 0 0]);
set(h,'LineWidth',2);

for k=1:length(ISet),
   h=line([JSet(k) jProj(k)],[ISet(k) iProj(k)]);
   set(h,'Color',[0 0 0]);
   set(h,'LineWidth',1);
end

xlabel(['PL: P1 = (' num2str(LimitI(1)) ', ' num2str(LimitJ(1))...
       '),   P2 = (' num2str(LimitI(2)) ', ' num2str(LimitJ(2)) ')'])

%==============================================================%
% End of file 'bestline.m'
```

```
function b=fillStraightLookUpFun(x)

% Description: Computes lookup table for use in detecting special
%              points of the edge image
%     1 4 7
% x   2 5 8
%     3 6 9

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'bifurcLookUpFun.m'

%-----------------------------------------------------------------%

% 'Y' configuration
p1=[0 1 0; 0 1 0;  1 0 1];
p2=rot90(p1);
p3=rot90(p2);
p4=rot90(p3);

% 'Y+' configuration
q1=[0 1 1; 0 1 0;  1 0 1];
q2=rot90(q1);
q3=rot90(q2);
q4=rot90(q3);

% 'Y++' configuration
r1=[1 1 1; 0 1 0;  1 0 1];
r2=rot90(r1);
r3=rot90(r2);
r4=rot90(r3);

% 'Y45' configuration
s1=[1 0 0; 0 1 1;  0 1 0];
s2=rot90(s1);
s3=rot90(s2);
s4=rot90(s3);

% 'T45' configuration
w1=[1 0 0; 0 1 0;  1 0 1];
w2=rot90(w1);
w3=rot90(w2);
w4=rot90(w3);

% 'T' configuration
z1=[1 1 1; 0 1 0; 0 1 0];
z2=rot90(w1);
```

```
z3=rot90(w2);
z4=rot90(w3);

% 'X' configuration
t1=[0 1 0; 1 1 1; 0 1 0];
t2=[1 0 1; 0 1 0; 1 0 1];

b=(sum(x(:)==p1(:))==9)|(sum(x(:)==p2(:))==9)|(sum(x(:)==p3(:))==9)|(sum(x(:)==
p4(:))==9)|...

(sum(x(:)==q1(:))==9)|(sum(x(:)==q2(:))==9)|(sum(x(:)==q3(:))==9)|(sum(x(:)==q4
(:))==9)|...

(sum(x(:)==r1(:))==9)|(sum(x(:)==r2(:))==9)|(sum(x(:)==r3(:))==9)|(sum(x(:)==r4
(:))==9)|...

(sum(x(:)==s1(:))==9)|(sum(x(:)==s2(:))==9)|(sum(x(:)==s3(:))==9)|(sum(x(:)==s4
(:))==9)|...

(sum(x(:)==w1(:))==9)|(sum(x(:)==w2(:))==9)|(sum(x(:)==w3(:))==9)|(sum(x(:)==w4
(:))==9)|...

(sum(x(:)==z1(:))==9)|(sum(x(:)==z2(:))==9)|(sum(x(:)==z3(:))==9)|(sum(x(:)==z4
(:))==9)|...
  (sum(x(:)==t1(:))==9)|(sum(x(:)==t2(:))==9);

%============================================================%
% End of file 'bifurcLookUpFun.m.m'
```

```
function [PLCluster, Clusterization, resolutionTouch, cornerTouch]=...
   breakPL(PL)
%
% [PLCluster, Clusterization, resolutionTouch, cornerTouch] =...
%    breakPL(PL)
%
% Description: Breaks the set of primitive line segments into
%              geografically unrelated clusters.
%
% PL column: [theta  d  base LimitI  LimitJ]'

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'breakPL.m'

n=size(PL,2);
resSeparationSquared = 8;  % 2^2 + 1^2 % =(2*sqrt(2))^2
resSeparation = sqrt(resSeparationSquared);

zerosnn=uint8(zeros(n,n));

Len=lengthOfPL(PL);
% GeoMedLenMatrix=sqrt((Len'*ones(1,n)) .* (ones(n,1)*Len));

% angleRelated=zerosnn;
perpRelated=zerosnn;
HALFANGLEMAX1=pi/8;
ComparisonAngle1=pi/4-HALFANGLEMAX1;
HALFANGLEMAX2=45*pi/(2*180); % 15 degrees / 2
ComparisonAngle2=pi/2-HALFANGLEMAX2;
% HALFANGLEMAX3=15*pi/(2*180); % 15 degrees / 2
% ComparisonAngle3=pi/2-HALFANGLEMAX3;

thetas=PL(1,:);
%DeltaThetaMatrix = thetas'*ones(1,n)-ones(n,1)*thetas;
%angleRelated(find(abs(mod(DeltaThetaMatrix,pi/2)-pi/4)>ComparisonAngle1))=1;
%angleRelated(find(eye(n)))=0; % exclude self
%perpRelated(find(abs(mod(DeltaThetaMatrix-pi/2,pi)-pi/2)>ComparisonAngle2))=1;

% To save memory, do it line-by-line:
for i=1:n,
   DeltaThetaMatrix = thetas-thetas(i);
   perpRelated(i,find(abs(mod(DeltaThetaMatrix-pi/2,pi)-
pi/2)>ComparisonAngle2))=1;
end
clear DeltaThetaMatrix
```

```
 [DummyDmin, DummyBestPair, d11] = ...
   distBetweenPoints(PL([5 7],:),PL([5 7],:));
d11(find(eye(n)))=Inf;
% touches11 = find(d11 <= GeoMedLenMatrix);
resTouches11 = find(d11 <= resSeparationSquared);
d11(find(~perpRelated))=Inf;
[minDistToAngleRelatedByColumn,RowIndexesWhereFound]=min(d11);

ColumnsWhereFound=find(minDistToAngleRelatedByColumn <
Len(RowIndexesWhereFound).*Len );

res11=zerosnn;
res11(resTouches11)=1;

cornerTouch11=[];
for c=1:length(ColumnsWhereFound),
   % test if both vertices are "alone", that is, not touching other PL
   if (sum(res11(:,ColumnsWhereFound(c)))==0)&...
         (sum(res11(RowIndexesWhereFound(ColumnsWhereFound(c)),:))==0)
      % if it is, then iclude it into cornerTouch class
      cornerTouch11 = [cornerTouch11...
         sub2ind([n
n],RowIndexesWhereFound(ColumnsWhereFound(c)),ColumnsWhereFound(c))'];
   end
end

clear res11 d11

[DummyDmin, DummyBestPair, d12] = ...
   distBetweenPoints(PL([5 7],:),PL([6 8],:));
% touches12 = find(d12 <= GeoMedLenMatrix);
resTouches12 = find(d12 <= resSeparationSquared);
d12(find(~perpRelated))=Inf;
[minDistToAngleRelatedByColumn,RowIndexesWhereFound]=min(d12);
%minDistToAngleRelatedByColumn(124)
%RowIndexesWhereFound(124)
%Len(RowIndexesWhereFound(124))*Len(124)

ColumnsWhereFound=find(minDistToAngleRelatedByColumn <
Len(RowIndexesWhereFound).*Len );

res12=zerosnn;
res12(resTouches12)=1;

cornerTouch12=[];
for c=1:length(ColumnsWhereFound),
   % test if both vertices are "alone", that is, not touching other PL
   if (sum(res12(:,ColumnsWhereFound(c)))==0)&...
         (sum(res12(RowIndexesWhereFound(ColumnsWhereFound(c)),:))==0)
      % if it is, then iclude it into cornerTouch class
      cornerTouch12 = [cornerTouch12...
         sub2ind([n
n],RowIndexesWhereFound(ColumnsWhereFound(c)),ColumnsWhereFound(c))'];
   end
end

clear res12 d12

[DummyDmin, DummyBestPair, d21] = ...
   distBetweenPoints(PL([6 8],:),PL([5 7],:));
```

```
% touches21 = find(d21 <= GeoMedLenMatrix);
resTouches21 = find(d21 <= resSeparationSquared);
d21(find(~perpRelated))=Inf;

[minDistToAngleRelatedByColumn,RowIndexesWhereFound]=min(d21);

ColumnsWhereFound=find(minDistToAngleRelatedByColumn <
Len(RowIndexesWhereFound).*Len );

res21=zerosnn;
res21(resTouches21)=1;

cornerTouch21=[];
for c=1:length(ColumnsWhereFound),
   % test if both vertices are "alone", that is, not touching other PL
   if (sum(res21(:,ColumnsWhereFound(c)))==0)&...
         (sum(res21(RowIndexesWhereFound(ColumnsWhereFound(c)),:))==0)
      % if it is, then iclude it into cornerTouch class
      cornerTouch21 = [cornerTouch21...
         sub2ind([n
n],RowIndexesWhereFound(ColumnsWhereFound(c)),ColumnsWhereFound(c))'];
   end
end

clear res21 d21

[DummyDmin, DummyBestPair, d22] = ...
   distBetweenPoints(PL([6 8],:),PL([6 8],:));
% touches22 = find(d22 <= GeoMedLenMatrix);
resTouches22 = find(d22 <= resSeparationSquared);
d22(find(~perpRelated))=Inf;

[minDistToAngleRelatedByColumn,RowIndexesWhereFound]=min(d22);

ColumnsWhereFound=find(minDistToAngleRelatedByColumn <
Len(RowIndexesWhereFound).*Len );

res22=zerosnn;
res22(resTouches22)=1;

cornerTouch22=[];
for c=1:length(ColumnsWhereFound),
   % test if both vertices are "alone", that is, not touching other PL
   if (sum(res22(:,ColumnsWhereFound(c)))==0)&...
         (sum(res22(RowIndexesWhereFound(ColumnsWhereFound(c)),:))==0)
      % if it is, then iclude it into cornerTouch class
      cornerTouch22 = [cornerTouch22...
         sub2ind([n
n],RowIndexesWhereFound(ColumnsWhereFound(c)),ColumnsWhereFound(c))'];
   end
end

clear res22 d22

cornerTouch =
union(union(cornerTouch11,cornerTouch12),union(cornerTouch21,cornerTouch22));
cornerRelated=zerosnn;
cornerRelated(cornerTouch)=1;

res11=zerosnn;
```

```
res12=zerosnn;
res21=zerosnn;
res22=zerosnn;
res11(resTouches11)=1;
res12(resTouches12)=1;
res21(resTouches21)=1;
res22(resTouches22)=1;
resolutionTouch = res11|res12|res21|res22;
clear res11 res12 res21 res22

D=(resolutionTouch)|(cornerRelated); % (obtuseRelated & resolutionTouch); %

p=etree(double(D));
ElimVector=p;


k=0;
PLCluster={};
Clusterization={};
Cluster=zeros(size(p));

i=1; j=1;
for i=1:n,
   if p(i)~=0,
      k=k+1;
      PLCluster{k}=PL(:,i);
      Clusterization{k}=[i];
      % Cluster(i)=k;
      j=i;
       while p(j)>0,
          PLCluster{k}=[PLCluster{k} PL(:,p(j))];
          Clusterization{k}=[Clusterization{k} p(j)];
          Cluster(j)=k;
          jOld=j;
          j=p(j);
          p(jOld)=0;
      end
      % if p(j) clusterized before, merge the two clusters:
      if Cluster(j)>0
         % exclude common before merging
         currentCount=length(Clusterization{k});
          PLCluster{k}=PLCluster{k}(:,1:currentCount-1);
          Clusterization{k}=Clusterization{k}(1:currentCount-1);

         PLCluster{Cluster(j)}=[PLCluster{Cluster(j)} PLCluster{k}];
         PLCluster=PLCluster(1:k-1);
         Clusterization{Cluster(j)}=...
            [Clusterization{Cluster(j)} Clusterization{k}];
         Cluster(Clusterization{k})=Cluster(j);
         Clusterization=Clusterization(1:k-1);
         k=k-1; % backup cluster counter
      else
         Cluster(j)=k;
      end
   end
end

%=================================================================%
% End of file 'breakPL.m'
```

```
function [MergedPL, ColinearIndexes, newMergedLines,...
   NumberOfClusters] = colinear(PL, A, cosColinearLimit, SINMAX)
%
% Description: Fuses colinar primitive segment lines that
%              are close to each other.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'colinear.m'

%------------------------------------------------------------------%

n=size(PL,2);

% limit value of differencial angle to be considered colinear:
% global cosColinearLimit

% limit value of vertex distance to be considered touching:
global limitDist;

thetas=PL(1,:);
cosDiffThetaInBetweenPL=...
   abs(triu((cos(triu(thetas'*ones(1,size(PL,2))...
     -ones(size(PL,2),1)*thetas,1))),1));

distanceParameters=PL(2,:);
   diffProjToCenter=  abs(triu(distanceParameters'*ones(1,size(PL,2))...
      -ones(size(PL,2),1)*distanceParameters));

MergedPL=PL;
newMergedLines=[];

% detect pairs of primitive lines that are approximately paralel
PairsOfParalelPL=find(cosDiffThetaInBetweenPL > cosColinearLimit);
disp([int2str(length(PairsOfParalelPL)) '/' int2str((n*n -n)/2)...
      ' (' num2str(round(1000*length(PairsOfParalelPL)/((n*n -n)/2))/10)...
      '%) pairs of paralel primitive lines found.']);

ColinearTouchingPairs=[];
if length(PairsOfParalelPL)>0
   PossiblePairs = PairsOfParalelPL(find(diffProjToCenter(PairsOfParalelPL) <
10));
   disp([int2str(length(PossiblePairs)) '/'
int2str(length(PairsOfParalelPL))...
      ' ('
num2str(round(1000*length(PossiblePairs)/(length(PairsOfParalelPL)))/10)...
       '%) pairs of possible primitive lines found.']);
```

```
    % PairsOfParalelPL = PossiblePairs; % these are paralel and not too far
(perp.)

    if length(PossiblePairs)>0
    [ColinearTouchingPairs,ColinearButNotNecessarilyTouchingPairs]...
        =findTouchingPairs(A, PL, PossiblePairs, n, limitDist, SINMAX);
        %
ColinearTouchingPairs=intersect(PairsOfParalelPL,favorablyClosePairs(PL,limitDi
st));
        disp([int2str(length(ColinearTouchingPairs)) '/'
int2str(length(PossiblePairs))...
            ' ('
num2str(round(1000*length(ColinearTouchingPairs)/(length(PossiblePairs)))/10)..
.
            '%) pairs of touching primitive lines found.']);
    end
end

ColinearIndexes=[];
if length(ColinearTouchingPairs)>0
    [L1, L2] = ind2sub([n n],ColinearTouchingPairs);
    selectedLines=zeros(1,n);
    selectedLines(L1)=1;
    selectedLines(L2)=1;
    ColinearIndexes=find(selectedLines);

    % cluster colinear pairs that touch each other
    S=clusterColinearTouchingPairs(ColinearTouchingPairs,n);
    NumberOfClusters=length(S);
    disp(['Num of Clusters Found: ' int2str(NumberOfClusters)]);

    unchangedList=ones(1,n);
    for i=1:NumberOfClusters,
        % disp(['Cluster #' int2str(i) ' = [' int2str(S{i}) ']'])
        [L1, L2] = ind2sub([n n],S{i});

         selectedLines=zeros(1,n);
         selectedLines(L1)=1;
         selectedLines(L2)=1;
        indexesOfLinesToBeMerged=find(selectedLines);
        ResultingLine=mergePrimitiveLines(PL(:,indexesOfLinesToBeMerged));
        sizeOfThisCluster=length(indexesOfLinesToBeMerged);
        thetaCluster = PL(1,indexesOfLinesToBeMerged);

        % only merge PL's if this cluster is fully connected
        % in the ColinearButNotNecessarilyTouchingPairs set
        AllConnections=[];
        for c1=1:length(L1),
           for c2=1:length(L2),
              if L1(c1) < L2(c2)
                 AllConnections=[AllConnections sub2ind([n n],L1(c1),L2(c2))];
              end
           end
        end

         unchangedList(indexesOfLinesToBeMerged)=0;
         newMergedLines=[newMergedLines ResultingLine];
    end
    disp([int2str(length(find(unchangedList))) ' PL remain unchanged.']);
```

```
   disp([int2str(size(newMergedLines,2)) ' merged PL replaced the clusters.']);

   MergedPL=[PL(:,find(unchangedList)) newMergedLines];
else
   NumberOfClusters=0;
end % if length(ColinearTouchingPairs)>0

%-----------------------------------------------------------------%
function b=LineInCommon(s,t,n)
%
[L1, L2] = ind2sub([n n],[s t]);
b=(L1(1)==L1(2))|(L1(1)==L2(2))|(L1(2)==L2(1))|(L2(1)==L2(2));

%-----------------------------------------------------------------%
function ResultingPL=mergePrimitiveLines(PL)
%
% pixel MSE version based on function 'bestline'
%
n=size(PL,2);
if isempty(PL)
   ResultingPL=PL;
else
   theta=PL(1,1);
   d=PL(2,1);
   base=PL(3:4,1);
   Center = base' + d*[cos(theta) sin(theta)];
   r=uint8(zeros(round(2*Center) + 1));

   for k=1:size(PL,2),
      theta=PL(1,k);
      d=PL(2,k);
      maxI=ceil(max(max(PL(5:6,:))));
      maxJ=ceil(max(max(PL(7:8,:))));
      LineLength=lengthOfPL(PL(:,k));
      for len=0:0.2:LineLength,
       pointNow=round([PL(5,k) PL(7,k)] + len*[-sin(theta) cos(theta)]);
         if (pointNow(1)<=maxI)&(pointNow(2)<=maxJ)&(1<=min(pointNow))
         r(pointNow(1),pointNow(2))=1;
       end
      end
   end
   ResultingPL = bestline(r,Center);
   % =[theta d base(1) base(2) LimitI(1) LimitI(2) LimitJ(1) LimitJ(2)]';
end

%-----------------------------------------------------------------%
function S=mergeClustersMarked(OldCluster,clustersToMerge);
% merge clusters marked:
% Eg.: From {S{1} S{2} S{3} S{4} S{5} S{6} S{7}}, marked [2 4 5]:
%       ==>  S{1}      S{3}            S{6} S{7} SNew,
%            SNew = S{2} U S{4} U S{5}
S={};
i=1;
mergedCluster=[];
for j=1:length(OldCluster),
   if isempty(find(clustersToMerge==j))
      S{i}=OldCluster{j};
      i=i+1;
   else
      mergedCluster=[mergedCluster OldCluster{j}];
```

```
    end
end
if ~isempty(mergedCluster)
   S{i}=sort(mergedCluster);
end

%----------------------------------------------------------------%
function S=clusterColinearTouchingPairs(ColinearTouchingPairs,n)
% cluster colinear pairs that touch each other

S{1}=ColinearTouchingPairs(1); % clusters: S{1}, S{2}, ...
for k=2:length(ColinearTouchingPairs),
   included=0;
   clustersToMerge=[];
   i=1;
   while i<=length(S), % test if pertain to any cluster
      j=1;
      touchingInThisCluster=0;
      while j<=length(S{i})&not(touchingInThisCluster),
         if LineInCommon(ColinearTouchingPairs(k),S{i}(j),n)
            touchingInThisCluster=1;
            % mark to merge the clusters
            clustersToMerge = [clustersToMerge i];
            if not(included) %
               S{i} = [S{i} ColinearTouchingPairs(k)];
               included=1;
            end
         end % if Touching(ColinearTouchingPairs(k),S{i}(j))

      j=j+1;
      end % while j<=length(S{i})&not(touchingInThisCluster)
      i=i+1;
   end % i<=length(S)

   if not(included)
      S{length(S)+1}=ColinearTouchingPairs(k);
   else
      S=mergeClustersMarked(S,clustersToMerge);
   end % not(included)
end % for k=2:length(ColinearTouchingPairs)

%----------------------------------------------------------------%
function [TouchingPairs, ColinearButNotNecessarilyTouchingPairs]...
   =findTouchingPairs(A, PL, PairsOfParalelPL, n, limitDist, SINMAX)
%
% find pairs of aligned PL that are enough close to each other
% by ONE of their extremities
%

R = 1;

limitDist2=limitDist*limitDist;
TouchingPairs=[];
ColinearButNotNecessarilyTouchingPairs=[];

[P, Q, LostPL] = pixelPL(PL,size(A));

for k=1:length(PairsOfParalelPL),

   % find lines L1, L2
   [L1, L2] = ind2sub([n n],PairsOfParalelPL(k));

   if ((L1==35) & (L2==40))
      flag=1;
   else
      flag=0;
   end

   % compute the pixels hit by the endpoints of L1 and L2
   [iP1, jP1] = integerEndPoints(PL, L1, size(A));
   [iP2, jP2] = integerEndPoints(PL, L2, size(A));

   % dij = distance(L1Pi, L2Pj)
   d11=sqrt((PL(5,L1)-PL(5,L2))*(PL(5,L1)-PL(5,L2)) + (PL(7,L1)-
PL(7,L2))*(PL(7,L1)-PL(7,L2)));
   d22=sqrt((PL(6,L1)-PL(6,L2))*(PL(6,L1)-PL(6,L2)) + (PL(8,L1)-
PL(8,L2))*(PL(8,L1)-PL(8,L2)));
   d12=sqrt((PL(5,L1)-PL(6,L2))*(PL(5,L1)-PL(6,L2)) + (PL(7,L1)-
PL(8,L2))*(PL(7,L1)-PL(8,L2)));
   d21=sqrt((PL(6,L1)-PL(5,L2))*(PL(6,L1)-PL(5,L2)) + (PL(8,L1)-
PL(7,L2))*(PL(8,L1)-PL(7,L2)));
   [dSort,sIndex]=sort([d11 d12 d22 d21]);
   mind=dSort(1);
   % min=d(i,i) => d(j,j) should be max;
   % min=d(i,j) => d(j,i) should be max, i,j in {1,2}
   otherIndex=mod((sIndex(1)-1)+2,4)+1;

   % compute which endpoint of L1 and L2 are the ones "touching" each other
   L1g = floor((sIndex(1)-1)/2) + 1;
   L2g = 1 + ((sIndex(1)==2)|(sIndex(1)==3));

   % find all other PL that have an endpoint in their neighborhood
   PLinNeighborhood = union(neighborPL(iP1(L1g), jP1(L1g), P, R),...
                            neighborPL(iP2(L2g), jP2(L2g), P, R));
   OtherPLinNeighborhood = setdiff(PLinNeighborhood, [L1 L2]);

   % only proceed with search if both conditions are met
   if (sIndex(4)==otherIndex)&isempty(OtherPLinNeighborhood)
     % disp(num2str(100*k/length(PairsOfParalelPL)));

     % posOK=' Good position ';

      % hij = perpendicular distance(Lj'Pi, Lj)
      h11 = sigdistoline(PL([5 7],L2)',PL(:,L1)); % dist from L1P2 to L1
      h22 = sigdistoline(PL([6 8],L1)',PL(:,L2)); % dist from L1P2 to L2
      h12 = sigdistoline(PL([5 7],L1)',PL(:,L2)); % dist from L1P1 to L2
      h21 = sigdistoline(PL([6 8],L2)',PL(:,L1)); % dist from L2P2 to L1

      Len1=lengthOfPL(PL(:,L1));
      Len2=lengthOfPL(PL(:,L2));

      L1withinL2=((h12*h22 <= 0)|((max(abs([h11 h22])) < limitDist))&...
%(Len2 > Len1)))&...
                    ((max(abs([h12/d11 h22/d21])) < SINMAX)|...
                    (max(abs([h12/d12 h22/d22])) < SINMAX));
      L2withinL1=((h11*h21 <= 0)|((max(abs([h11 h21])) < limitDist))&...
%(Len1 > Len2)))&...
                    ((max(abs([h11/d11 h21/d12])) < SINMAX)|...
                    (max(abs([h11/d21 h21/d22])) < SINMAX));
      if L1withinL2|L2withinL1
         ColinearButNotNecessarilyTouchingPairs...
            =[ColinearButNotNecessarilyTouchingPairs PairsOfParalelPL(k)];
         if mind < min([Len1 Len2])
            TouchingPairs = [TouchingPairs PairsOfParalelPL(k)];
          end % if (mind < min([Len1 Len2]))
      end % if L1withinL2|L2withinL1
   else
      % posOK=' Bad position ';
   end % if (sIndex(4)==otherIndex)
end % for k=1:length(PairsOfParalelPL),

%-----------------------------------------------------------------%
```

function pairs = favorablyClosePairs(PL,limitDist)

```
n=size(PL,2);
limitDist2=limitDist*limitDist;
%
Delta11I=ones(n,1)*PL(5,:) - PL(5,:)'*ones(1,n);
Delta11J=ones(n,1)*PL(7,:) - PL(7,:)'*ones(1,n);
D11=Delta11I.*Delta11I + Delta11J.*Delta11J;
% rem: sqrt(D11(k,k)) = 0, D11 symetric

Delta22I=ones(n,1)*PL(6,:) - PL(6,:)'*ones(1,n);
Delta22J=ones(n,1)*PL(8,:) - PL(8,:)'*ones(1,n);
D22=Delta22I.*Delta22I + Delta22J.*Delta22J;
% rem: sqrt(D12(k,k)) = 0, D22 symetric

Delta12I=ones(n,1)*PL(5,:) - PL(6,:)'*ones(1,n);
Delta12J=ones(n,1)*PL(7,:) - PL(8,:)'*ones(1,n);
D12=Delta12I.*Delta12I + Delta12J.*Delta12J;
% rem: sqrt(D12(k,k)) = ||L(k)|| D12 potentially not symetric

D=zeros(n,n,4);
D(:,:,1)=D11;
D(:,:,2)=D12;
D(:,:,3)=D22;
D(:,:,4)=D12';
[minD,minIndex]=min(D,[],3);
[maxD,maxIndex]=max(D,[],3);

otherIndex=mod((minIndex-1)+2,4)+1;

pairs=intersect(find(minD<limitDist2),find((maxIndex-otherIndex)==0));

%-----------------------------------------------------------------%
% Debug functions
```

function debugShowCluster(S)

```
%
str='S = {';
for k=1:length(S),
   str=[str ' [' int2str(S{k}) ']'];
end
disp([str ' }'])


%=================================================================%
% End of file 'colinear.m'
```

```
function b=cornerLookUpFun(x)
%
%     1 4 7
% x   2 5 8
%     3 6 9
%
% Description: Computes lookup table for detection of corners
%              in the edge image.
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'cornerLookUpFun.m'

%------------------------------------------------------------------%

b=(sum(x(:)==[0 0 0  1 1 0  0 1 0]')==9)|(sum(x(:)==[0 1 0  1 1 0  0 0
0]')==9)|...
  (sum(x(:)==[0 1 0  0 1 1  0 0 0]')==9)|(sum(x(:)==[0 0 0  0 1 1  0 1
0]')==9)|...
  (sum(x(:)==[1 0 0  0 1 0  1 0 0]')==9)|(sum(x(:)==[1 0 1  0 1 0  0 0
0]')==9)|...
  (sum(x(:)==[0 0 1  0 1 0  0 0 1]')==9)|(sum(x(:)==[0 0 0  0 1 0  1 0
1]')==9);


%==================================================================%
% End of file 'cornerLookUpFun.m'
```

```
function createCornerLookUp
%
% Description: Creates in memory lookup tables for
%              detection of special points in edge image.
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'createCornerLookUp.m'

%------------------------------------------------------------------%

global CornerLookUpTable BifurcLookUpTable FillStraightGapsLookUpTable

CornerLookUpTable=makelut('cornerLookUpFun',3);

BifurcLookUpTable=makelut('bifurcLookUpFun',3);

FillStraightGapsLookUpTable=makelut('fillStraightLookUpFun',3);

%==================================================================%
% End of file 'createCornerLookUp.m'
```

```
function [dmin, bestPair, d] = distBetweenPoints(PA,PB)
%
% PA (2 x m) and PB (2 x n) are arrays of points.
%
% Description: Computes the distance between points of two sets
%              of points A & B. For every point Ai in A end Bj in
%              B, a distance d(ij) will be computed. dmin is the
%              minimum of these distances, obtained at the best
%              pair (i, j).
%

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'distBetweenPoints.m'

%----------------------------------------------------------------%

m=size(PA,2);
n=size(PB,2);
d=zeros(m,n);
d=inf;

DI=PA(1,:)'*ones(1,n)-ones(m,1)*PB(1,:);
DJ=PA(2,:)'*ones(1,n)-ones(m,1)*PB(2,:);
% d=sqrt(DI.*DI + DJ.*DJ);
d = DI.*DI + DJ.*DJ;

[dClusterAtoEachB, Indexes] = min(d);
[dmin, Jmin] = min(dClusterAtoEachB);
dmin=sqrt(dmin);
Imin = Indexes(Jmin);
bestPair=[Imin Jmin];

%================================================================%
% End of file 'distBetweenPoints.m'
```

```
function [E, CB, sel] = edgedetec(A)
%
% Description: enhanced edge detection & edge split points
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Albérto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'edgedetec.m'

%------------------------------------------------------------------%

global CornerLookUpTable BifurcLookUpTable FillStraightGapsLookUpTable

MainContourAnalysis=0;

if MainContourAnalysis
   [Contour, sel, threshold] = maincontours(A);

   for k=1:length(threshold),
   % E=edge(A,'canny');
      Aux=zeros(size(A));
   Aux(find(A>=threshold(k)))=1;
      E{k}=edge(Aux,'canny');

      % E=E|applylut(E,FillStraightGapsLookUpTable);
      E{k}=bwmorph(E{k},'clean');
   Corners=applylut(E{k},CornerLookUpTable);
      Bifurcs=applylut(E{k},BifurcLookUpTable);
      CB{k}=Corners|Bifurcs;
   end
else
   [E,th]=edge(A,'canny');
   E=edge(A,'canny',[th(1) max([th(1) th(2)/2])]);
   E=bwmorph(E,'clean');
   Corners=applylut(E,CornerLookUpTable);
   Bifurcs=applylut(E,BifurcLookUpTable);
   CB=Corners|Bifurcs;
   E={E};
   CB={CB};
   sel=1;
end

%==================================================================%
% End of file 'edgedetec.m'
```

```
function b=fillStraightLookUpFun(x)
%     1 4 7
% x   2 5 8
%     3 6 9
%
% Description: Computes lookup table for finding special points
%              in the edge image.
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'fillStraightLookUpFun.m'

%-----------------------------------------------------------------%

p1=[0 0 0; 1 0 1;  0 0 0];
p2=rot90(p1);

q1=[1 0 0; 0 0 0;  0 0 1];
q2=rot90(q1);

b=(sum(x(:)==p1(:))==9)|(sum(x(:)==p2(:))==9)|...
   (sum(x(:)==q1(:))==9)|(sum(x(:)==q2(:))==9);

%=================================================================%
% End of file 'fillStraightLookUpFun.m'
```

```
function [P, Indexes] = fPartition(S)
%
% function [P, Indexes] = fPartition(S)
%
% S={S(i)}
%
% Description:  Eliminates sets S(i) in the partion S,
%               if there is some S(j) contained in S(i)

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'fPartition.m'

%------------------------------------------------------------------%

n=length(S);
EmptyList=uint8(zeros(1,n));
DiscardMark=uint8(zeros(1,n));
SearchList=uint8(ones(1,n));
for k=1:n,
   if isempty(S{k})
      SearchList(k)=0;
      EmptyList(k)=1;
   end
end

i=1;
while i < n,
   if ~EmptyList(i)
      j = 1;
      while j <= n,
      if SearchList(j)&(i~=j)
          if prod(ismember(S{i},S{j}))==1
          DiscardMark(j)=1;
          SearchList(i)=0;
             end
      end
          j = j + 1;
      end
   end
   i = i + 1;
end;
Indexes = find(~DiscardMark);
P = S(Indexes);

%==================================================================%
% End of file 'fPartition.m'
```

```
function similar = fuzzyeq(LineDescription, TotalLineDescription)
%
% Use:
%
%      similar = fuzzyeq(LineDescription,TotalLineDescription)
%
% where
%
%     LineDescription = [angle, disp, baseI, baseJ,...
%                        LimitI1, LimitI2, LimitJ1, LimitJ2]
%
% Description: Checks if there is a similar line segment in
%              'TotalLineDescription' to 'LineDescription'

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'fuzzyeq.m'

%-----------------------------------------------------------------%

global limitDist
angle=LineDescription(1);
disp=LineDescription(2);
base=LineDescription(3:4)';
LimitI=LineDescription(5:6)';
LimitJ=LineDescription(7:8)';

similar=0;
% k=size(TotalLineDescription,2);
k=1;
while (k <= size(TotalLineDescription,2))&not(similar),
   a=TotalLineDescription(1,k);
   d=TotalLineDescription(2,k);
   b=TotalLineDescription(3:4,k)';
   LI=TotalLineDescription(5:6,k)';
   LJ=TotalLineDescription(7:8,k)';
   D1=sqrt((LI(1)-LimitI(1))*(LI(1)-LimitI(1)) + (LJ(1)-LimitJ(1))*(LJ(1)-
LimitJ(1)));
   D2=sqrt((LI(2)-LimitI(2))*(LI(2)-LimitI(2)) + (LJ(2)-LimitJ(2))*(LJ(2)-
LimitJ(2)));
   similar = (max([D1 D2]) < limitDist);
   k=k+1;
end

%=================================================================%
% End of file 'fuzzyeq.m'
```

```
function  G = graphPL(PL, P, IJ, sizeA)
%
% G = graphPL(PL, P, IJ, sizeA)
%
% Description: Computes the endpoint connectivity graph.
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'graphPL.m'

%-----------------------------------------------------------------%

n=size(PL,2);

G=uint8(zeros(2*n, 2*n));

R = 2;
radiusSquared = (R + 0.5)*(R + 0.5);
neighborhoodMask = uint8(zeros(2*R+1,2*R+1,size(P,3)));
for i=-R:1:R,
   for j=-R:1:R,
      if (i*i + j*j) <= radiusSquared
         neighborhoodMask(i+R+1,j+R+1,:)=1;
      end
   end
end

% connect the endpoints of the same line-segment
for k=1:n,
   P1 = 2*(k-1) + 1;
   P2 = 2*(k-1) + 2;
   G(P1, P2) = 1;
   G(P2, P1) = 1;
end

% connect the neighbor endpoint s
for k=1:n,
   [iP, jP] = integerEndPoints(PL, k , sizeA);

   for g=1:2,
      i = iP(g);  j = jP(g);

      NeighborhoodIRange = [max([1 i-R]):min([i+R sizeA(1)])];
      NeighborhoodJRange = [max([1 j-R]):min([j+R sizeA(2)])];

      onBorders = (i < R+1)|(j < R+1)|(i+R > sizeA(1))|(j+R > sizeA(2));
```

```
        NeighborhoodP = P(NeighborhoodIRange,NeighborhoodJRange,:);

       if onBorders
          % raw squared neighborhood
           [I, J, H] = ind2sub([length(NeighborhoodIRange)
length(NeighborhoodJRange)],...
             find(NeighborhoodP(:,:,:)));
       else
          % refined circular neighborhood
           [I, J, H] = ind2sub([length(NeighborhoodIRange)
length(NeighborhoodJRange)],...
             find(NeighborhoodP(:,:,:)&neighborhoodMask));
       end

        for t=1:length(I), % test all vertices found inside Neighborhood
          endPointNUmber = NeighborhoodP(I(t), J(t), H(t));
          if G(2*(k-1)+g, endPointNUmber)==0
             G(2*(k-1)+g, endPointNUmber)=2;
          end
       end
       G(2*(k-1)+g, 2*(k-1)+g)=0;
    end
end

zerosnn=uint8(zeros(n,n));
Len=lengthOfPL(PL);
perpRelated=zerosnn;
paraRelated=zerosnn;
% transvRelated=uint8(zeros(2*n,2*n));
HALFANGLEMAX1=20*pi/(2*180); % 15 degrees / 2
ComparisonAngle1=pi/2-HALFANGLEMAX1;
HALFANGLEMAX2=45*pi/(2*180); % 15 degrees / 2
ComparisonAngle2=pi/2-HALFANGLEMAX2;
ComparisonAngle3=pi/4-HALFANGLEMAX1;
thetas=PL(1,:);
% To save memory, do it line-by-line:
for i=1:n,
   DeltaThetaMatrix = thetas-thetas(i);
   perpRelated(i,find(abs(mod(DeltaThetaMatrix-pi/2, pi) - pi/2) >
ComparisonAngle2))=1;
   paraRelated(i,find(abs(mod(DeltaThetaMatrix, pi) - pi/2) >
ComparisonAngle1))=1;
end
clear DeltaThetaMatrix

for k1=1:n,
   % k1
   for k2=k1+1:n,
      %if (k1== ) & (k2== )
      % [k1 k2]
      %end

      k = [k1 k2];

      % connect the closer endpoints of perpendicular line-segments
       if perpRelated(k1, k2)
          [dmin, bestPair, d] = ...
             distBetweenPoints([PL([5 7], k1) PL([6 8], k1)],[PL([5 7], k2)
PL([6 8], k2)]);
```

```
          if dmin < sqrt(prod(Len([k1 k2])))
             p1 = 2*(k1-1) + bestPair(1);
             p2 = 2*(k2-1) + bestPair(2);
             if isempty(find(G(p1,:)==2)) | isempty(find(G(p2,:)==2))
                % plotwithlines(zeros(sizeA),{PL(:,[k1 k2])}, 2, {[0 1 0]});
pause

                G(p1, p2) = 3;
             G(p2, p1) = 3;
             end
          end
       end

       % connect the closer endpoints of aligned parallel line-segments
       if paraRelated(k1, k2)
          [lenMin, whoLenMin] = min(Len([k1 k2]));
          [lenMax, whoLenMax] = max(Len([k1 k2]));

          [dmin, bestPair, dSquared] = ...
             distBetweenPoints([PL([5 7], k1) PL([6 8], k1)],[PL([5 7], k2)
PL([6 8], k2)]);

          oppositePair = 3 - bestPair;

          if (dmin < lenMin)&(dSquared(oppositePair(1),oppositePair(2)) <
lenMin*lenMin)
             p1 = 2*(k1-1) + bestPair(1);
             p2 = 2*(k2-1) + bestPair(2);
             if isempty(find(G(p1,:)==2)) | isempty(find(G(p2,:)==2))
                PLAux = PLfromPoints(IJ(p1,:), IJ(p2,:), sizeA);
                if abs(mod(PL(1,k(whoLenMax))-PLAux(1),pi/2)-pi/4) >
ComparisonAngle3
                   % plotwithlines(zeros(sizeA),{PL(:,[k1 k2])}, 2, {[0 1 0]});
pause
                   G(p1, p2) = 5;
                G(p2, p1) = 5;
                end
             end
             p1 = 2*(k1-1) + oppositePair(1);
             p2 = 2*(k2-1) + oppositePair(2);

             if isempty(find(G(p1,:)==2)) | isempty(find(G(p2,:)==2))
                PLAux = PLfromPoints(IJ(p1,:), IJ(p2,:), sizeA);
                if abs(mod(PL(1,k(whoLenMax))-PLAux(1),pi/2)-pi/4) >
ComparisonAngle3
                % plotwithlines(zeros(sizeA),{PL(:,[k1 k2])}, 2, {[0 1 0]});
pause
                   G(p1, p2) = 5;
                G(p2, p1) = 5;
                end
             end
          end
       end

   end
end

for k1=1:n,
   % k1
   for k2=k1+1:n,
        k = [k1 k2];

        % connect the endpoint if touching body of perpendicular line-segment
        if perpRelated(k1, k2)
           proj=zeros(2,2);

           [h(1), proj(1,:)] = sigdistoline(IJ(2*(k2-1) + 1,:), PL(:, k1));
            [h(2), proj(2,:)]= sigdistoline(IJ(2*(k2-1) + 2,:), PL(:, k1));
           [minh, gmin] = min(abs(h));
            if (abs(h(gmin)) <  2*sqrt(2)) & (prod(h) >= 0) &...
                   (abs(distBetweenPoints(proj(gmin,:)',PL([5 7],k1))+...
                       distBetweenPoints(proj(gmin,:)',PL([6 8],k1))-Len(k1)) < 1)
              q1 = 2*(k1-1) + 1;
              q2 = 2*(k1-1) + 2;
              p = 2*(k2-1) + gmin;
              % if (G(p, q1)==0)&(G(p, q2)==0)
              if (sum(G(p, q1)==[0 3])==1)&(sum(G(p, q2)==[0 3])==1)
                 G(p, q1) = 4;
                 G(p, q2) = 4;
                 G(q1, p) = 4;
                 G(q2, p) = 4;
              end
              % plotwithlines(zeros(sizeA),{PL(:,[k1 k2])}, 2, {[1 0 0]}); pause
           end

           [h(1), proj(1,:)] = sigdistoline(IJ(2*(k1-1) + 1,:), PL(:, k2));
            [h(2), proj(2,:)] = sigdistoline(IJ(2*(k1-1) + 2,:), PL(:, k2));
            [minh, gmin] = min(abs(h));
            if (abs(h(gmin)) <  2*sqrt(2)) & (prod(h) >= 0) &...
               (abs(distBetweenPoints(proj(gmin,:)',PL([5 7],k2))+...
                   distBetweenPoints(proj(gmin,:)',PL([6 8],k2))-Len(k2)) < 1)

              q1 = 2*(k2-1) + 1;
              q2 = 2*(k2-1) + 2;
              p = 2*(k1-1) + gmin;
              if (sum(G(p, q1)==[0 3])==1)&(sum(G(p, q2)==[0 3])==1)
                 G(p, q1) = 4;
                 G(p, q2) = 4;
                 G(q1, p) = 4;
                 G(q2, p) = 4;

                 %cancelLinks = find(G(p,:)==3);
                 %G(p,cancelLinks)=0;
                 %G(cancelLinks,p)=0;
              end
              % plotwithlines(zeros(sizeA),{PL(:,[k1 k2])}, 2, {[1 0 0]}); pause
           end
        end
    end
end
clear perpRelated paraRelated

%==================================================================%
% End of file 'graphPL.m'
```

```
function [ISeq, JSeq, totalPerimeter, supportedFraction]...
   = IJSeqFromPath(loopPath, IJCoordinates, PL, G, sizeA)
%
% [ISeq, JSeq] = IJSeqFromPath(loopPath, IJCoordinates, G, sizeA)
%
% Description: Computes the polygon determined by a cycle in G.

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'IJSeqFromPath.m'

%----------------------------------------------------------------%

c = loopPath([1 2]);

for k=3:length(loopPath),
   p2 = loopPath(k);
   p1 = 2*floor((p2-1)/2) + 2 - mod(p2-1, 2);
   c = [c p1 p2];
end
c = [c loopPath([1 2])];

% Len = lengthOfPL(PL);

ISeq = IJCoordinates(c(1:2),1)';
JSeq = IJCoordinates(c(1:2),2)';
supportedPerimeter = distBetweenPoints([ISeq(1) JSeq(1)]', [ISeq(2) JSeq(2)]');
totalPerimeter = supportedPerimeter;

k=3;
while k <= length(c)-1,
   % p = 2*floor((p2-1)/2) + 2 - mod(p2-1, 2);
   gJump = double(G(c(k-1),c(k)));
   switch gJump
      case {1,2,5}
         ISeq = [ISeq IJCoordinates(c(k),1)];
         JSeq = [JSeq IJCoordinates(c(k),2)];
         jumpLength = distBetweenPoints(IJCoordinates(c(k-1),:)',
IJCoordinates(c(k),:)');
         totalPerimeter = totalPerimeter + jumpLength;
         switch gJump
            case 1
               supportedPerimeter = supportedPerimeter + jumpLength;
            case {2,5}
         end
      case 2222 % (perimeter computation is approximated)
```

```
         PL1=PLfromPoints(IJCoordinates(c(k-2),:),IJCoordinates(c(k-
1),:),sizeA);
         PL2=PLfromPoints(IJCoordinates(c(k),:),IJCoordinates(c(k+1),:),sizeA);
         if abs(cos(mod(PL1(1)-PL2(2)-pi/2,pi))) > cos(pi/6)
            intersectionPoint = ...
            intersectLines(IJCoordinates([c(k-2) c(k-1)],:),...
               IJCoordinates([c(k) c(k+1)],:));
            ISeq(length(ISeq)) = intersectionPoint(1);
            JSeq(length(JSeq)) = intersectionPoint(2);
         else
            ISeq = [ISeq IJCoordinates(c(k),1)];
            JSeq = [JSeq IJCoordinates(c(k),2)];
         end
         jumpLength = distBetweenPoints(IJCoordinates(c(k-1),:)',
IJCoordinates(c(k),:)');
         totalPerimeter = totalPerimeter + jumpLength;

      case 3
         intersectionPoint = ...
            intersectLines(IJCoordinates([c(k-2) c(k-1)],:),...
                           IJCoordinates([c(k) c(k+1)],:));
         K = length(ISeq)+1;
         previousJump = distBetweenPoints([ISeq(K-2) JSeq(K-2)]', [ISeq(K-1)
JSeq(K-1)]');
         jumpLength = distBetweenPoints([ISeq(K-2) JSeq(K-2)]',
intersectionPoint');
         totalPerimeter = totalPerimeter + jumpLength - previousJump;
         if jumpLength < previousJump
            supportedPerimeter = supportedPerimeter + jumpLength -
previousJump;
         end

         jumpAfterCorner = distBetweenPoints(IJCoordinates(c(k+1),:)',
intersectionPoint');
         totalPerimeter = totalPerimeter + jumpAfterCorner;

         supportedPerimeter = supportedPerimeter + ...
            min([jumpAfterCorner,...
               distBetweenPoints(IJCoordinates(c(k),:)',
IJCoordinates(c(k+1),:)')]);

         ISeq(length(ISeq)) = intersectionPoint(1);
         JSeq(length(JSeq)) = intersectionPoint(2);

         if k < length(c)-1
            % do nothing
         else
            firstJump = distBetweenPoints([ISeq(1) JSeq(1)]', [ISeq(2)
JSeq(2)]');
            totalPerimeter = totalPerimeter - firstJump;
            supportedPerimeter = supportedPerimeter - firstJump;

            ISeq = ISeq(2:length(ISeq));
            JSeq = JSeq(2:length(JSeq));
            ISeq = [ISeq ISeq(1)];
            JSeq = [JSeq JSeq(1)];
         end

      case 4
         intersectionPoint = ...
```

```
              intersectLines(IJCoordinates([c(k-2) c(k-1)],:),...
              IJCoordinates([c(k) c(k+1)],:));

         K = length(ISeq)+1;

         previousJump = distBetweenPoints([ISeq(K-2) JSeq(K-2)]', [ISeq(K-1)
JSeq(K-1)]');

         jumpLength = distBetweenPoints([ISeq(K-2) JSeq(K-2)]',
intersectionPoint');
         totalPerimeter = totalPerimeter + jumpLength - previousJump;

         if jumpLength < previousJump
            supportedPerimeter = supportedPerimeter + jumpLength -
previousJump;
         end

         jumpAfterCorner = distBetweenPoints(IJCoordinates(c(k+1),:)',
intersectionPoint');
         totalPerimeter = totalPerimeter + jumpAfterCorner;

          supportedPerimeter = supportedPerimeter + ...
            min([jumpAfterCorner,...
               distBetweenPoints(IJCoordinates(c(k),:)',
IJCoordinates(c(k+1),:)')]);

          ISeq(length(ISeq)) = intersectionPoint(1);
          JSeq(length(JSeq)) = intersectionPoint(2);

          if k < length(c)-1
             ISeq = [ISeq IJCoordinates(c(k),1)];
         JSeq = [JSeq IJCoordinates(c(k),2)];
         else
            firstJump = distBetweenPoints([ISeq(1) JSeq(1)]', [ISeq(2)
JSeq(2)]');
            totalPerimeter = totalPerimeter - firstJump;
            supportedPerimeter = supportedPerimeter - firstJump;
            ISeq = ISeq(2:length(ISeq));
            JSeq = JSeq(2:length(JSeq));
            ISeq = [ISeq ISeq(1)];
            JSeq = [JSeq JSeq(1)];
         end
      otherwise
         disp('')
   end
   k = k + 1;
end

supportedFraction = supportedPerimeter / totalPerimeter;

%------------------------------------------------------------------%
function P = intersectLines(L1, L2);
P1=L1(1,:); Q1=L1(2,:);
P2=L2(1,:); Q2=L2(2,:);
u1=Q1-P1; u2=Q2-P2;
L = inv([u1' u2'])*(P2-P1)';
P = P1 + L(1)*u1; % = P2 + L(2)*u2;

%==================================================================%
% End of file 'IJSeqFromPath.m'
```

```
function [CX, CY, V]=improfile2(A,JVSeq,IVSeq)
%
% Description: Find the sequence of pixels along a polygonal line,
%              given by its vertices.

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'improfile2.m'

%----------------------------------------------------------------%

[m,n]=size(A);
controlFrame=zeros(m,n);

CX=[];
CY=[];
V=[];
dist=zeros(1,3);
for k=1:length(JVSeq)-1,
   P0=min(max(round([IVSeq(k)   JVSeq(k)  ]-0.5), 1),[m n]);
   P1=min(max(round([IVSeq(k+1) JVSeq(k+1)]-0.5), 1),[m n]);
   controlFrame(P0(1),P0(2))=1;
   CY=[CY P0(1)];
   CX=[CX P0(2)];
   V=[V A(P0(1),P0(2))];
   PNow=P1;
   while sum(PNow==P0)~=2,
      controlFrame(PNow(1),PNow(2))=1;
      CY=[CY PNow(1)];
      CX=[CX PNow(2)];
      V=[V A(PNow(1),PNow(2))];
      DI=sign(P0(1) - PNow(1));
      DJ=sign(P0(2) - PNow(2));
      Q=[(PNow+[DI DJ])' (PNow+[DI 0])' (PNow+[0 DJ])'];

      % on the line, N = ||(j - j0)*(i - i1) - (j - j1)*(i - i0)|| = 0
      for q=1:3,
         dist(q)=abs((Q(2,q)-P0(2))*(Q(1,q)-P1(1)) - (Q(2,q)-P1(2))*(Q(1,q)-
P0(1)));
      end
      [dmin,indexToNewP]=min(dist);
      PNow=Q(:,indexToNewP)';
   end
end

%================================================================%
% End of file 'improfile2.m'
```

```
function  [iP, jP] = integerEndPoints(PL, k , sizeA)
%
% Description: Builds a table with the coordinates of the endpoints,
%              rounded to the nearest integer.

%==============================================================%
%                                                              %
% COMPUTER-AIDED RECOGNITION OF                                %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                    %
%                                                              %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe %
%                                                              %
% Department of Computer Science                               %
% Naval Postgraduate School, September 1999                    %
%                                                              %
%==============================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'integerEndPoints.m'

%--------------------------------------------------------------%

iP = round(PL([5 6],k)');
jP = round(PL([7 8],k)');
iP = max(iP, [1 1]);
jP = max(jP, [1 1]);
iP = min(iP,sizeA(1)*[1 1]);
jP = min(jP,sizeA(2)*[1 1]);

%==============================================================%
% End of file 'integerEndPoints.m'
```

```
function c = lengthOfPL(PL)
%
% Description: computes the length of primitive line segments in 'PL'
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'lengthOfPL.m'

%------------------------------------------------------------------%

DI = PL(5,:)-PL(6,:);
DJ = PL(7,:)-PL(8,:);
c = sqrt(DI.*DI + DJ.*DJ);

%==================================================================%
% End of file 'lengthOfPL.m'
```

```
function booleanReturn = lineseg(Seg)
%
% Description: Defines criterion for acceptable edge segments.
%

%==============================================================%
%                                                              %
% COMPUTER-AIDED RECOGNITION OF                                %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                    %
%                                                              %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe %
%                                                              %
% Department of Computer Science                               %
% Naval Postgraduate School, September 1999                    %
%                                                              %
%==============================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:     'lineseg.m'

%--------------------------------------------------------------%


% Edges should have at least three pixels.

booleanReturn = length(find(Seg))>=3;


%==============================================================%
% End of file 'lineseg.m'
```

```
function [loop, h] = loopfromPL(InitialPath, HMAX, IJCoordinates, A, PL);
%
%  Usage:
%
%  [loop, h] = loopfromPL(InitialPath, HMAX, IJCoordinates, A, PL);
%
%  Description: Searches for a cycle in G containing 'IntialPath'.
%
%  Global G is nxn binary matrix representing the edge
%  connections in an oriented graph of N vertices.
%  "p", one of the vertices; "H" max depth of search.
%
%
%
%                         P=P0
%                     /    |    \
%                    /     |     \
%                 P11    P12    P13
%                 /          /  |  \
%               P21        P22 P23 P24
%                           /   |
%                         P31  P32=P0  =>   {P0, P13, P23, P0}
%                                              cycle found!
%
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'loopfromPL.m'

global G

debugOn = 0;

N = size(G,1);

f = InitialPath(1);
p = InitialPath(length(InitialPath));

prohibited = InitialPath(2:length(InitialPath));

for k=1:N,
   distance(k) = sum((IJCoordinates(k,:) - IJCoordinates(f,:)).^2);
end

Found = 0;
Fail = 0;
```

```
counter = ones(1,HMAX);
cMax = zeros(1,HMAX);

h = 1;
nextSet = 1;
while (~Found)&(~Fail),
   f = InitialPath(1);
   p = InitialPath(length(InitialPath));

   pathSet = InitialPath(1:length(InitialPath)-1);
   prohibited = InitialPath(2:length(InitialPath));
   nextSet = 1;
   h = 1;

   % place to probe counter, if debugging

   if debugOn
   plotwithlines(A,{PL PL(:,floor((prohibited-1)/2)+1) PL(:,floor((pathSet-
1)/2)+1) },[2 2 2],{[1 0 1][1 0 0][0 1 0]})
      pause
   end

   while (~isempty(nextSet)) & (h <= HMAX) & (~Found) & (~Fail),
      [nextSet, pathSet, prohibited, Found] = nextTo(p, pathSet, prohibited,
f);

      % debug only
      cMax(h) = length(nextSet);

      % distance from nextSet(k) to f
      [dummySorted, sIndex] = sort(distance(nextSet));
      nextSet = nextSet(sIndex);

      if counter(h) > length(nextSet)
         if h > 1
            counter(h-1) = counter(h-1) + 1;
            counter(h:HMAX)=1;
            h = 1;
            p = InitialPath(length(InitialPath));
            pathSet = InitialPath(1:length(InitialPath)-1);
             prohibited = InitialPath(2:length(InitialPath));
         else
            Fail = 1;
         end
      else

       if (~isempty(nextSet))&(h < HMAX)
            p = nextSet(counter(h));

            %
            if debugOn
               v=axis;
               plotwithlines(A,{PL PL(:,floor((prohibited-1)/2)+1)...
                 PL(:,floor((pathSet-1)/2)+1) PL(:,floor((p-1)/2)+1)},...
                    [2 2 2 2],{[1 0 1][1 0 0][0 1 0][1 1 0]})
               str=[];
               for ih=1:h,
               str=[str int2str(pathSet(1+ih)) ':(' int2str(counter(ih))...
                   '/' int2str(cMax(ih)) '), '];
                 end
                   str = [str '-->' int2str(p) '  options: '...
                       int2str(find(G(pathSet(1+h),:)>1)) ' types: '...

int2str(double(G(pathSet(1+h),find(G(pathSet(1+h),:)>1))))];
                title(str)
                xlabel(int2str(pathSet))
                 axis(v)
                   pause
                end
             h = h + 1;

            else
               counter(h) = counter(h) + 1;
               counter(h:HMAX)=1;
            end
        end
    end
end

if Found
   loop = pathSet;
else
   loop = [];
end

%==================================================================%
% End of file 'loopfromPL.m'
```

```
function PLinNeighborhood = neighborPL(i, j, P, R)
%
%  Usage:
%         PLinNeighborhood = neighborPL(i, j, P, R)
%
%  Description:
%  Finds all the indexes of all primitive line-segments that
%  have endpoints in the R-radius neighborhood of (i,j), by
%  inspecting the endpoint lookup table 'P'.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'neighborPL.m'

sizeA = size(P);

PLinNeighborhood = [];

NeighborhoodIRange = [max([1 i-R]):min([i+R sizeA(1)])];
NeighborhoodJRange = [max([1 j-R]):min([j+R sizeA(2)])];

NeighborhoodP = P(NeighborhoodIRange,NeighborhoodJRange,:);

[I, J, H] = ind2sub([length(NeighborhoodIRange) length(NeighborhoodJRange)],...
   find(NeighborhoodP(:,:,:)));

for t=1:length(I), % test all vertices found inside Neighborhood
   endPointNUmber = NeighborhoodP(I(t), J(t), H(t));
   PLnumber = floor((endPointNUmber-1)/2)+1;
   PLinNeighborhood = [PLinNeighborhood PLnumber];
end

PLinNeighborhood = unique(PLinNeighborhood);

%==================================================================%
% End of file 'neighborPL.m'
```

```
function [nextSet, pathSet, prohibited, Found] = nextTo(i, path, prohibited, f)
%
%  Usage:
%          [nextSet, pathSet, prohibited, Found]...
%                  = nextTo(i, path, prohibited, f)
%
%  Description:
%          Evaluates the next possibilities for path continuation
%          from the current 'path', when searching for cycles.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'nextTo.m'

global G
next = setdiff(find(G(i,:) > 1), prohibited);
prohibited = [prohibited next];

Found = ismember(f, next);%&isempty(intersect(next, setdiff(path,f)));
if Found
   nextSet = f; % next;
   pathSet = [path i];
else
   if ~isempty(next)
      for k=1:length(next),
         next(k) = find(G(next(k),:)==1);
      end
      next = setdiff(next, prohibited);
      prohibited = [prohibited next];
   end
    if isempty(next)
    nextSet = [];
       pathSet = [];
    Found = 0;
   else
      if length(next)==1
         [nextSet, pathSet, prohibited, Found] = ...
            nextTo(next, [path i], prohibited, f);
       else
          nextSet = next;
          pathSet = [path i];
      end
    end
end

%==================================================================%
% End of file 'nextTo.m'
```

```
%
% Description:
%
%  This script program:
%  (i) Computes the (Canny's method)edge image from the input image.
%  (ii) Detects some of the corners and junctions for enhanced
%       line-segment extraction by morphological operations.
%

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PhaseI.m'

%----------------------------------------------------------------%

disp('======================================================================')
;
disp(['Begining of edge extraction phase for image in: ''' FileName '''']);
T1=clock; [B, CB, sel] = edgedetec(A); T2=clock;
timeSpentExtractingEdges=etime(T2,T1);
disp(['Edge extraction completed: ET=' num2str(timeSpentExtractingEdges)]);

% Compute pixels in any main edge
BTotal=zeros(size(B{1}));
for k=1:length(sel),
   BTotal=B{k}|BTotal;
end

% plot edges extracted from 'A'
if showContours
   fig1handle=figure(1);
   for k=1:length(sel),
      EdgeOnlyImage=...
         uint8(round(255*(1-double(BTotal))));
      % imagesc(EdgeOnlyImage);
      imwrite(EdgeOnlyImage,[FileName '.edgeOnly(' int2str(k)...
         'of' int2str(length(sel)) ').tif'],'tif');

      EdgeWithCornersImage=...
         uint8(round(255*(21 - 16*double(CB{k})-4*double(B{k})-
double(BTotal))/21));
      imagesc(EdgeWithCornersImage);
      imwrite(EdgeWithCornersImage,[FileName '.edgeWithCorners('...
         int2str(k) 'of' int2str(length(sel)) ').tif'],'tif');
      colormap(gray);
      title([''''' FileName ''': Edge extraction ' int2str(k)...
         ' of ' int2str(length(sel))]);
```

```
    axis image
    axis on
       if logResults
          hgsave(fig1handle,[FileName '.edge(' int2str(k) 'of'...
             int2str(length(sel)) ').Fig' ]);
       else
          pause(1)
       end
    end
end

%==================================================================%
% End of file 'PhaseI.m'
```

```
%
% Description: This script program extracts primitive lines from
%              the edge image derived from original input image.
%              Results are plot graphically.
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational`System:  Windows NT 4.0
%
% This file named:    'PhaseII.m'

%-----------------------------------------------------------------%

disp('==========================================================================')
;
disp(['Begining of primitive line search phase for image in: ''' FileName
'''']);
fig2handle=figure(2);
T1=clock; PL = prilines(A, B, CB, sel); T2=clock;
timeSpentExtractingPL=etime(T2,T1);

% plot primitive lines extracted from 'A'
clf
plotwithlines(A,{PL},1.5,{[0 0 1]});
title([int2str(size(PL,2)) ' primitives line segments found']);
xlabel([date ' ' int2str(T2(4)) ':' sprintf('%2.2d',T2(5))...
   ',  ET=' num2str(round(10*etime(T2,T1))/10) 's'])
disp(['End of primitive line search phase: ET=' num2str(etime(T2,T1))]);

if logResults
   hgsave(fig2handle,[FileName '.PL.Fig']);
   save([FileName '.PhaseII.mat']);
end

%=================================================================%
% End of file 'PhaseII.m'
```

```
%
% Description: This script program clusters the line segements
%              extracted from the original image in approximately
%              unrelated sets, to break the complexity of the
%              connectivity analysis to follow. Then plots the
%              resulting clusters with a number of different colors
%              for improved visualization. The subprogram that
%              actually computes the clustering is 'breakPL',
%              called once from this code.
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PhaseIIA.m'

%------------------------------------------------------------------%

PLTotal=PL;
[PLCluster, Clusterization] = breakPL(PLTotal);
PLClusterOrig=PLCluster;
ClusterizationOrig=Clusterization;

SizeCluster=[];
for w=1:size(ClusterizationOrig,2),
   SizeCluster=[SizeCluster size(ClusterizationOrig{w},2)];
end
[SortedSizeCluster,IndexSorted]=sort(SizeCluster);
SortedSizeCluster = fliplr(SortedSizeCluster);
IndexSorted = fliplr(IndexSorted);

for w=1:size(ClusterizationOrig,2),
   PLCluster{w}=PLClusterOrig{IndexSorted(w)};
   Clusterization{w}=ClusterizationOrig{IndexSorted(w)};
end

SeparatingColor=[...
   1 1 0   2;... %yellow
   0 1 0 1.5;... %green
   0 0 1 1.5;... %blue
   1 0 0 1.5;... %red
   1 0.5 0 2;... %orange
   0 0.5 0 2;... %dark green
   0 1   1 2;... %cyan
   0.5 0.5 0 2;  ... %dark brown
   0.75 0.75 0 2;... %brown
   0.5  0    1 2;... %purple
   1    0    1 2;... %magenta
   1 0.75 0.75 2;... %pink
   0.6 0.6   1 2;... %light blue
];

S=0;
thickNessOfColor=zeros(1,size(Clusterization,2));
for w=1:size(Clusterization,2),
   colors{w}=SeparatingColor(mod(w-1,size(SeparatingColor,1))+1,1:3);
   thickNessOfColor(w)=...
      SeparatingColor(mod(w-1,size(SeparatingColor,1))+1,4);
   S=S+size(Clusterization{w},2);
end

fig3handle=figure(3);
plotwithlines(A,PLCluster,thickNessOfColor,colors);
title([int2str(S) ' out of ' int2str(size(PL,2))...
      ' PL were clustered into ' int2str(size(Clusterization,2))...
      ' sets. Largest cluster has '...
      int2str(size(PLCluster{1},2)) ' PL.']);

if logResults
   hgsave(fig3handle,[FileName '.PLCluster.Fig']);
   save([FileName '.PhaseIIA.mat']);
end

%==============================================================%
% End of file 'PhaseIIA.m'
```

```
%
% Description: This script program merges approximately colinear
%              primitive line segments.
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PhaseIII.m'

%-----------------------------------------------------------------%

disp('-------------------------------------------------------------');
disp(['Begining of PL merge phase for image in: ''' FileName '''']);

MergedPL1 = {};
NewLines = [];

T1=clock;
for k=1:length(PLCluster),
   disp(['Merging PL cluster ' int2str(k) ' in: ''' FileName '''']);
   [Merged1, ColinearIndexes1, NewMergedLines1, NumberOfClusters1] =...
      colinear(PLCluster{k}, A, cos(20*pi/180), sin(5*pi/180));
   MergedPL1{k} = Merged1;
   NewLines = [NewLines NewMergedLines1];

   if (~logResults) & (size(MergedPL1{k},2) > 2)

      % plot primitive lines extracted from 'A', emphasized colinear touching
lines

      fig3Ahandle=figure(4);


      plotwithlines(A,{PLCluster{k} PLCluster{k}(:,ColinearIndexes1)...
         PLCluster{k}(:,ColinearIndexes1)},...
            [1 3 2],{[0 0 1][1 0 0][1 1 0]});
      title([int2str(length(ColinearIndexes1)) ' aligned PL''s found in 2nd
step.'])

       fig3Bhandle=figure(5);
         plotwithlines(A,{PLCluster{k} PLCluster{k}(:,ColinearIndexes1)...
            NewMergedLines1 NewMergedLines1},...
               [3 2 3 2],{[1 0 0][1 1 0][1 1 0][0 0 0]});

      title(['PL merging step 1: Number of PL''s reduced to '
int2str(size(MergedPL1{k},2)) '.'])
       xlabel([date ' ' int2str(T2(4)) ':' sprintf('%2.2d',T2(5))...
                ',  ET=' num2str(round(10*etime(T2,T1))/10) 's'])

        pause(1)
    end
end
T2=clock;

disp(['End of PL merge phase, step 1: ET=' num2str(etime(T2,T1))]);

fig3Chandle=figure(6);
plotwithlines(uint8(255*ones(size(A))),{[MergedPL1{:}] NewLines NewLines},...
    [2 3 2],{[0 0 1][1 0 0][1 1 0]});

title([int2str(size(NewLines,2)) ' merged lines, remaining '...
    int2str(size([MergedPL1{:}],2)) ' PL']);

MergedPL=MergedPL1;

disp(['Total PL merge phase: ET=' num2str(etime(T2,T1))]);

if logResults
    hgsave(fig3Chandle,[FileName '.MergedPL.fig']);
    save([FileName '.PhaseIII.mat']);
end

%==================================================================%
% End of file 'PhaseIII.m'
```

```
function [PL, loop, PLinLoop, DecomposedPLLoops, Indexes,...
   contourLoop, supportedFraction, shaperErr, errMax, IJCoordinates] = ...
   PhaseIVA(A, unsortedPL, logResults, FileName, PLClusterNumber)

%
%  loop{k} = [sequence of endpoint s] defining a closed path in G,
%              starting from node 2k and primitive line segment k
%              (second endpoint in the sequence is node 2k-1, for
%              which we have G(2k,2k-1)=1).
%              If a cycle is not found at the maximum depth of
%              search adopted and starting from line segment k,
%              loop{k} will be empty. Following the two first
%              end-nodes in loop{k} that belong to the same PL,
%              only the 'leaving' end-node of each PL will be
%              represented. Thus if a cycle is formed by x PL,
%              length(loop{k}) will be  2 + (x-1) = x+1
%
%  PLinLoop{k} = [sorted set of indexes of those PL forming loop{k}]
%
%  DecomposedPLLoops = the cycles that don't contain cycles
%
%  Indexes = the indexes in loop and PLinLoop for those cycles that don't
%            contain cycles
%
%  contourLoop{i} = matriz mx2 of IJCoordinates of the polygon associated
%                   with the i-th cycle that don't contain cycles
%
%
%  Description: Performs the connectivity analysis on graph G,
%               finding the cycles, computing buiding-likelihood
%               indexes for each of them and plotting the results
%               graphically.
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PhaseIVA.m'

%-----------------------------------------------------------------%

global G

% Debug shortcut
RecomputeG = 1;
if RecomputeG
   Len=lengthOfPL(unsortedPL);
   [sortedLen, sLenIndex]=sort(Len);
   PL=unsortedPL(:, fliplr(sLenIndex));

   [P, Q, LostPL, IJCoordinates] = pixelPL(PL, size(A));

   disp(['Building endpoints graph for cluster #'...
      int2str(PLClusterNumber) ' of ''' FileName '''']);

   G = graphPL(PL, P, IJCoordinates, size(A));
end

fig4handle=figure(7);

debugFlag = 0;

% find cycles in graph G with distance sort depth-first algorithm
[loop, PLinLoop, h] = ...
   smartFindCycles(G, A, PL, IJCoordinates, debugFlag);

if length([loop{:}]) > 0
   % eliminate cycles that contain cycles
   [DecomposedPLLoops, Indexes] = ...
      seploops(A, PLinLoop, PL, 1, debugFlag);

   if length(Indexes) > 0
      % compute error measures and optionally plot for debugging
      [contourLoop, shaperErr, errMax, supportedFraction] = ...
        plotcontours(A, loop, PLinLoop, G, Indexes,...
         PL, IJCoordinates, debugFlag);
   else
      contourLoop = [];
      shaperErr = [];
      errMax = [];
      supportedFraction = [];
   end
else
   DecomposedPLLoops = [];
   Indexes = [];
   contourLoop = [];
   shaperErr = [];
   errMax = [];
   supportedFraction = [];
end

%===============================================================%
% End of file 'PhaseIVA.m'
```

```
function [NumberOfBuildingClusters, BuildingCluster] = ...
   PhaseVA(A, imageBackground, Building, PLCluster,...
   pixelLength, Filename, debugMode, numberPlot)
%
% Building =
%               PL: {1:NumberOfBuildingCycles}
%            Cycle: {1:NumberOfBuildingCycles}
%     OwnerCluster: [1:NumberOfBuildingCycles]
%
% Description: Assembles the building clusters from the polygons
%              that are building contour candidates and plots them
%              with different saturated random colors, to
%              improve visualization.

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PhaseVA.m'

%-----------------------------------------------------------------%

clf reset

if numberPlot
   if imageBackground
      imagesc(0.75 + 0.25*double(A)/255, [0 1]); axis image; ...
         colormap(gray); plotColor=[1 1 1]*0.9;
   else
      image(uint8(255*ones(size(A)))); axis image; ...
         colormap(gray); plotColor=[1 1 1]*0.9;
   end
else
   if imageBackground
      imagesc(double(A)/255, [0 1]); axis image; ...
         colormap(gray); plotColor=[0 1 0];
   else
      image(uint8(255*ones(size(A)))); axis image;...
         colormap(gray); plotColor=[0 0 0];
   end
end

if iscell(PLCluster)
   NumberOfClusters=length(PLCluster);
else
   NumberOfClusters=1;
end

% Detect touching building cycles and form building clusters
```

```
NumberOfBuildingClusters = 0;
BuildingCluster.Cluster = [];
BuildingCluster.OwnerCluster = [];
BuildingCluster.IndexesInCluster = [];

for k=1:NumberOfClusters,
   % find all the cycles in the k-th cluster of primitive lines
   if iscell(PLCluster)
      whoInCluster = find(Building.OwnerCluster==k);
   else
      whoInCluster = [1:size(PLCluster,2)];
   end

   if ~isempty(whoInCluster)
      % merge those cycles who have non-null intersection
      [Clusters, PartitionIndexes] = ...
         rPartition(Building.PL(whoInCluster));

      for i=1:length(PartitionIndexes),

      BuildingCluster.IndexesInCluster = ...
            [BuildingCluster.IndexesInCluster...
               {whoInCluster(PartitionIndexes{i})}];
      end

      NumberOfBuildingClusters = ...
         NumberOfBuildingClusters + length(PartitionIndexes);

      BuildingCluster.Cluster = ...
         [BuildingCluster.Cluster Clusters];

      BuildingCluster.OwnerCluster = ...
         [BuildingCluster.OwnerCluster...
            k*ones(1,length(PartitionIndexes))];
   end
end

BuildingCluster.ICenter = zeros(1,NumberOfBuildingClusters);
BuildingCluster.JCenter = zeros(1,NumberOfBuildingClusters);
BuildingCluster.Area = zeros(1,NumberOfBuildingClusters);
BuildingCluster.AvLight = zeros(1,NumberOfBuildingClusters);
BuildingCluster.StdDevLight = zeros(1,NumberOfBuildingClusters);


for i=1:length(BuildingCluster.IndexesInCluster),
   BuildingCluster.Area(i) = 0;
   tColor = 2*pi*rand;
   RandomColor = (1+[cos(tColor) cos(tColor+2*pi/3)...
      cos(tColor+4*pi/3)])/2;
   RandomColor2 = (1+[cos(tColor+pi) cos(tColor+2*pi/3+pi)...
      cos(tColor+4*pi/3 + pi)])/2;

   areaControlFrame=uint8(zeros(size(A)));

   if ~isempty(BuildingCluster.IndexesInCluster{i})
      for j=1:length(BuildingCluster.IndexesInCluster{i}),
         controlFrame=uint8(zeros(size(A)));

         ISeq = Building.Contour{BuildingCluster.IndexesInCluster{i}(j)}.ISeq;
      JSeq = Building.Contour{BuildingCluster.IndexesInCluster{i}(j)}.JSeq;

      [CX, CY, C] = improfile2(A,JSeq,ISeq);
      onBorders{j} = unique(sub2ind(size(A),CY,CX));
      controlFrame(onBorders{j})=1;

      areaControlFrame = areaControlFrame | bwfill(controlFrame,'holes');

       % imagesc(controlFrame)

      if numberPlot
         patch(JSeq, ISeq, plotColor);
      else
         patch(JSeq, ISeq, RandomColor);
      end
      % pause, for debugging
   end

   areaIncludingBordersInPixels = sum(areaControlFrame(:));

   [IPixels, JPixels] = ind2sub(size(A), find(areaControlFrame));
   BuildingCluster.ICenter(i) = sum(IPixels)/areaIncludingBordersInPixels;
   BuildingCluster.JCenter(i) = sum(JPixels)/areaIncludingBordersInPixels;

   edgeAreaControlFrame = edge(areaControlFrame,'canny');

    % Debug patch (uncomment for debugging):

   % clf
   % subplot(121); imagesc(areaControlFrame); colormap(1-gray/10); axis
image;
   % subplot(122); imagesc(edgeAreaControlFrame); colormap(1-gray/10); axis
image;
   % pause

   perimeterInPixels = sum(edgeAreaControlFrame(:));

   areaEstimate = (pixelLength^2)*...
      (areaIncludingBordersInPixels - perimeterInPixels/2);
   BuildingCluster.Area(i) = areaEstimate;

   innerPixels = setdiff(find(areaControlFrame(:)),...
      find(edgeAreaControlFrame(:)));
   BuildingCluster.AvLight(i)=...
      sum(double(A(innerPixels)))/length(innerPixels);
   BuildingCluster.StdDevLight(i)=std(double(A(innerPixels)));

    if numberPlot
      h=text(BuildingCluster.JCenter(i),...
         BuildingCluster.ICenter(i),int2str(i));
      set(h,'Color',[0 0 0],'FontWeight','bold');
   else
      plotcross(BuildingCluster.JCenter(i),...
         BuildingCluster.ICenter(i),[1 1 1]);
   end

   if debugMode
      h1=line([1 340],[BuildingCluster.ICenter(i)...
         BuildingCluster.ICenter(i)]);
      h2=line([BuildingCluster.JCenter(i)...
            BuildingCluster.JCenter(i)],[1 260]);

        pause
           delete(h1)
          delete(h2)
       end

   end
end

% print target table - building clusters
tableTitle = ...
     ['|   Building Cluster List for Image in ''
|'];
tableTitle(41:41+length(Filename))=[Filename ''''];
disp('+-------------------------------------------------------------------------
+');
disp(tableTitle)
disp('+-----------+---------+---------+------------+---------+--------------
+');
disp('| Target ID | Coord I | Coord J |  Area (m2) |  Av Lum |  Std Dev Lum |')
disp('+-----------+---------+---------+-------------------------------------
+');
for i=1:length(BuildingCluster.IndexesInCluster),
   disp(['|    ' sprintf('%05d',i) '    | ' ...
          sprintf('%7.1f',BuildingCluster.ICenter(i))...
          ' | ' sprintf('%7.1f',BuildingCluster.JCenter(i)) ' |     '...
          sprintf('%7.0f',BuildingCluster.Area(i)) ' | '...
           sprintf('%7.1f',BuildingCluster.AvLight(i)) ' |      '...
           sprintf('%7.1f',BuildingCluster.StdDevLight(i)) ' |'...
       ])
end
disp('+-----------+---------+---------+-------------------------------------
+');
```

function plotcross(J, I, color)

```
d=0.75;
L=2;

JSeq=[J-d-L   J-d   J-d   J+d   J+d   J+d+L  J+d+L...
   J+d  J+d  J-d  J-d  J-d-L  J-d-L];
ISeq=[I-d    I-d    I-d-L I-d-L I-d    I-d    I+d ...
   I+d  I+d+L I+d+L I+d   I+d I-d];

patch(JSeq, ISeq, color)

%=================================================================%
% End of file 'PhaseVA.m'
```

```
function [P, Q, LostPL, IJCoordinates] = pixelPL(PL, sizeA)
%
% [P, Q, LostPL, IJCoordinates] = pixelPL(PL, sizeA)
%
% up to 4 PL vertices may coincide at pixel
%
% Description: Computes lookup tables for the endpoints of line
%              segments. The tables are used to speed up
%              computation of which line segments are in the
%              neighborhood of a given point. Up to four endpoints
%              are allowed to coincide on the same pixel. The
%              fifth and those beyond are lost (what is very
%              unlikely to happen).
%

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'pixelPL.m'

%-----------------------------------------------------------------%

n=size(PL,2);

IJCoordinates = zeros(n,2);

LostPL=[];
P=zeros(sizeA(1),sizeA(2),4);
Q=zeros(sizeA(1),sizeA(2),4);
for k=1:n,
   [iP, jP] = integerEndPoints(PL, k, sizeA);

   inserted=[0 0];

   for g=1:2,
      IJCoordinates(2*(k-1) + g,:) = PL([5 7]+g-1,k)';
      h=0;
      while (h < 4)&(~inserted(g)),
      h=h+1;
       if P(iP(g),jP(g),h)==0,
            P(iP(g),jP(g),h)=2*(k-1) + g;
            if jP(1)~=jP(2)
               Q(iP(g),jP(g),h)=mod(PL(1,k)+(g-1)*pi,2*pi);
            else
               if iP(1) < iP(2)
                  Q(iP(1),jP(1),h)= 3*pi/2;
                  Q(iP(2),jP(2),h)= pi/2;
               else
```

```
                  Q(iP(1),jP(1),h)= pi/2;
                  Q(iP(2),jP(2),h)= 3*pi/2;
               end
            end
            inserted(g)=1;
      end
      end
   end

   if prod(inserted)~=1
      LostPL = [LostPL k];
   end

end

%==================================================================%
% End of file 'pixelPL.m'
```

```
function PL = PLfromPoints(P,Q,sizeA)
%
% Description: Creates line segment parametric description from two
%              non-coincident points.
%

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'PLfromPoints.m'

%------------------------------------------------------------------%

LimitI=[P(1) Q(1)];
LimitJ=[P(2) Q(2)];
[LimitJ, Indexes]=sort(LimitJ);
LimitI=LimitI(Indexes);

theta=atan2(LimitI(1)-LimitI(2),LimitJ(2)-LimitJ(1));

% compute origin
CenterXY = floor((sizeA+1)/2);
CenterR=[1+sizeA(1)-CenterXY(1) CenterXY(2)]  - [0.5 0.5];

% compute base point, closest point to the origin on the line
Disp=CenterR - P(:)';
Y=-Disp(1);
X=Disp(2);
YSinXCos=Y*sin(theta) + X*cos(theta);
XProj=cos(theta)*YSinXCos;
YProj=sin(theta)*YSinXCos;
Base=P+[-YProj XProj];

% compute distance to center
d=sign(double(Base(1) < CenterR(1))-0.5)*norm(Base-CenterR);

PL = [theta d Base LimitI LimitJ]';

%==================================================================%
% End of file 'PLfromPoints.m'
```

```
function [contourLoop, error, errMax, supportedFraction] =...
   plotcontours(A, loop, PLinLoop, G, Indexes, PL,...
    IJCoordinates, debugMode)

% Description: Computes the building-likelihood indexes and
%              plots the cycles corresponding to the most likely
%              building contours.

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'plotcontours.m'

%-----------------------------------------------------------------%

Len = lengthOfPL(PL);

contourLoop = {};

imagesc(A); colormap(gray); axis image
for k=1:length(Indexes),
   %imagesc(A); colormap(gray); axis image

    for m=1:length(Indexes{k}),
      [ISeq, JSeq, totalPerimeter,...
         supportedFraction{length(Indexes{k})*(k-1) + m}] =...
          IJSeqFromPath(loop{Indexes{k}(m)},...
                IJCoordinates, PL, G, size(A));
      contourLoop{(length(Indexes{k})*(k-1) + m)}=[ISeq' JSeq'];

      PLIndexes = unique(floor((loop{Indexes{k}(m)}-1)/2)+1);
       [sLenSel, sIndexes]=sort(Len(PLIndexes));
       sIndexes = fliplr(sIndexes);
       BasePL = PL(:,PLIndexes(sIndexes(1)));
       BaseTheta = BasePL(1);

      [error{(length(Indexes{k})*(k-1) + m)},...
         errMax{(length(Indexes{k})*(k-1) + m)},...
          error2{(length(Indexes{k})*(k-1) + m)}]...
                = quadError(ISeq, JSeq, BaseTheta, size(A), 0);

      if debugMode
         plotwithlines(A,{PL PL(:,PLinLoop{Indexes{k}(m)})},...
            [3 3],{[0 0 1][1 0 0]})
         title([int2str(Indexes{k}(m)) ': ['...
            int2str(loop{Indexes{k}(m)}) '], shapErr='...
            num2str(error{(length(Indexes{k})*(k-1) + m)})...
            ', sFrac=' num2str(supportedFraction{length(Indexes{k})*(k-1) +
m})...
            ', maxErr=' num2str(errMax{length(Indexes{k})*(k-1) + m}) ])
       xlabel([' I:' int2str(IJCoordinates(loop{Indexes{k}(m)},1)')...
                '  J:' int2str(IJCoordinates(loop{Indexes{k}(m)},2)')])
         ylabel(int2str(length(PLinLoop{Indexes{k}(m)})))
       h=line(JSeq,ISeq);
       set(h,'LineWidth',1)
       set(h,'Color',[0 1 0])
      end
   end
   if debugMode
       pause
   end
end

for k=1:length(Indexes),

   for m=1:length(Indexes{k}),

      if (((length(PLinLoop{Indexes{k}(m)})<=4)&...
        (error{length(Indexes{k})*(k-1) + m} < 0.40))|...
       ((supportedFraction{length(Indexes{k})*(k-1) + m}>0.85)&...
           (error{length(Indexes{k})*(k-1) + m} < 0.20)))

      ISeq = contourLoop{(length(Indexes{k})*(k-1) + m)}(:,1);
         JSeq = contourLoop{(length(Indexes{k})*(k-1) + m)}(:,2);

         h=line(JSeq,ISeq);
         set(h,'LineWidth',2)
          set(h,'Color',[0 1 0])
      % pause(1)
      end
   end
   % pause
end

%================================================================%
% End of file 'plotcontours.m'
```

```
function plotwithlines(A, PrimitiveLines, thickness, colors);
%
% Description: Plots primitive line segments extracted from
%              image 'A' supperposed on 'A'. The set of
%              line segments may be partioned into clusters,
%              situation where colors and thicknesses can
%              be individually speciafied for each cluster.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'plotwithlines.m'

%------------------------------------------------------------------%

imagesc(double(A)/255, [0 1]); axis on; axis image; colormap (gray)
hold on

if (size(PrimitiveLines,2)>1)
   if (length(colors)==1)
      propColor = colors{1};
   colors=[];
      for k=1:size(PrimitiveLines,2)
      colors{k} = propColor;
      end
   end
   if (length(thickness)==1)
      thickness = thickness*ones(1,size(PrimitiveLines,2));
   end
end

for lineSet=1:length(PrimitiveLines),
   for lineSegIndex=1:size(PrimitiveLines{lineSet},2),
   LimitI=PrimitiveLines{lineSet}(5:6,lineSegIndex)';
      LimitJ=PrimitiveLines{lineSet}(7:8,lineSegIndex)';
      d=0.25; %1+thickness(lineSet)/2;
      patch([LimitJ(1)-d LimitJ(1)+d LimitJ(1)+d LimitJ(1)-d LimitJ(1)-d],...
            [LimitI(1)-d LimitI(1)-d LimitI(1)+d LimitI(1)+d LimitI(1)-d],...
            colors{lineSet})
      h=line(LimitJ,LimitI);
      set(h,'Color',colors{lineSet});
      set(h,'LineWidth',thickness(lineSet));
   end
end
hold off

%==================================================================%
% End of file 'plotwithlines.m'
```

```
function [TotalLineDescription] = prilines(A, B, CB, sel)
%
% Description: Primitive line segments extraction with
%              the use of the Radon Transform.
%
%    [B, sel, TotalLineDescription] = prilines(A)
%
%    'A' is a MxN matrix representing a gray level image
%
%    Each 'B{k}' is an edge images extracted from 'A', for
%    k in the range [1:length(sel)]
%
%    'CB{k}' is a MxN binary image with CB{k}(i,j)=1
%    where a possible corner was morphologically
%    extracted from 'A', k in the range [1:length(sel)]
%
%    'sel' is a list of indexes to the edge sets extracted
%
%    Each column of 'TotalLineDescription' describes a
%    primitive line extracted from 'A'.

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'prilines.m'

%-----------------------------------------------------------------%

ThetaRange=[0:179];
TotalLineDescription = [];

% Image segmentation on main edge images
for k=1:length(sel),
   ContourLineDescription = [];
   CenterXY = floor((size(B{1})+1)/2);
   CenterRBig=[1+size(B{1},1)-CenterXY(1) CenterXY(2)]  - [0.5 0.5];

   Rest=B{k}&(~CB{k});

   Seg=zeros(size(B{k}));
   SegNumber = 0;

   % figure

   while ~(max(max(Rest))==0),
      % imagesc(Rest,[0 1]);colormap(1-gray);axis image;title('Rest'); pause

      TBeginSeg=clock;
```

```
[Seg, Rest] = segm(Rest,8);
SegCopy=Seg;
% imagesc(Seg,[0 1]);colormap(1-gray);axis image;title('Seg Now');pause

AreaSeg=bwarea(Seg);

SegNumber = SegNumber+1;

% crop around the segment, saparing a one-pixel border
% around it for possible intersection with CB{k}:

[ISet, JSet] = ind2sub(size(Seg),find(Seg>0));
minValI=min(ISet);
minValI=max([minValI-1 1]);

maxValI=max(ISet);
maxValI=min([maxValI+1 size(A,1)]);

minValJ=min(JSet);
minValJ=max([minValJ-1 1]);

maxValJ=max(JSet);
maxValJ=min([maxValJ+1 size(A,2)]);

auxSeg=Seg|CB{k};
transfSeg=auxSeg(minValI:maxValI,minValJ:maxValJ);

% compute origin of small frame
CenterXYSmall = floor((size(transfSeg)+1)/2);
CenterRSmall=[1+size(transfSeg,1)-CenterXY(1) CenterXY(2)]  - [0.5 0.5];
CenterRSmallInBigCoord = CenterRSmall + [minValI-1 minValJ-1];

CPUT2=cputime;
[R, Xp]=radon(transfSeg,ThetaRange);
CPUT1=cputime;
% disp(['Radon transform time: CPU=' num2str(CPUT1-CPUT2)])

[maxRforEachTheta, maxIndex] = max(R);
[maxR, thetaIndex] = max(maxRforEachTheta);
globalMaxR=maxR;
dispIndex=maxIndex(thetaIndex);
% maxR=max(max(R));

radonMin = 2*3/4;

if maxR > radonMin,
   LPDetected=0;

   [mask, CenterR, theta, d, base] = ...
      radsel(size(transfSeg), size(R), thetaIndex, dispIndex);

   bigMask = zeros(size(Seg));
   bigMask(minValI:maxValI,minValJ:maxValJ)=mask;

   dummyPoint=[0 0];
   d = d + sigdistoline(CenterRSmallInBigCoord,...
      [theta 0 CenterRBig dummyPoint dummyPoint]');
```

```
          r = bigMask.*double(auxSeg); % B{k};

          CPUT2=cputime;
          [UsedPixels, LineDescription] = xlines(r, theta, d);
          CPUT1=cputime;

          %disp(['XLINES time: CPU=' num2str(CPUT1-CPUT2)]);
          % only include line primitive if not similar to any previously
detected
          for newL=1:size(LineDescription,2),
             Line=LineDescription(:,newL);

             seenBefore = fuzzyeq(Line,TotalLineDescription);

             % [newL seenBefore]
              if not(seenBefore)
                ContourLineDescription = [ContourLineDescription Line];
                TotalLineDescription = [TotalLineDescription Line];

                LPDetected=LPDetected+1;
                SegCopy(UsedPixels{newL})=0;
             end
          end
          if LPDetected>0
             Rest = Rest | SegCopy;
          end
       end % (maxR > radonMin)

       TEndSeg=clock;
       disp(['==> Contour #' int2str(sel(k)) ', LP Extraction #'
int2str(SegNumber)...
                 ', ' int2str(LPDetected) ' LP detected, ET='
num2str(etime(TEndSeg, TBeginSeg))]);
        %  disp(' ')

   end % while ~(max(max(Rest))==0)

end % for k=1:length(sel)

%==================================================================%
% End of file 'prilines.m'
```

```
function [errorAv, errorMax, errorAvRef] = ...
   quadError(ISeq, JSeq, BaseTheta, sizeA, debugMode)
%
% Description: Computes the deviations from ideal shapes for
%              building contours.

%=================================================================%
%                                                                 %
% COMPUTER-AIDED RECOGNITION OF                                   %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                       %
%                                                                 %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe   %
%                                                                 %
% Department of Computer Science                                  %
% Naval Postgraduate School, September 1999                       %
%                                                                 %
%=================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'quadError.m'

%-----------------------------------------------------------------%

errorTotal = 0;
lengthTotal = 0;
errorMax = 0;

global P

LenAux = zeros(1,length(ISeq)-1);
errorInJump = zeros(1,length(ISeq)-1);
thetaAux = zeros(1,length(ISeq)-1);

if debugMode
    figure(8)
   image(uint8(255*ones(sizeA))); colormap(gray); axis image
end

for k=2:length(ISeq),
   PLAux = PLfromPoints([ISeq(k-1) JSeq(k-1)],...
      [ISeq(k) JSeq(k)], sizeA);
   thetaAux(k-1) = PLAux(1);
   LenAux(k-1) = lengthOfPL(PLAux);
   errorInJump(k-1) = LenAux(k-1)*abs(sin(2*(mod(thetaAux(k-1)...
      - BaseTheta, pi/2))));
   errorTotal  = errorTotal + errorInJump(k-1);
   lengthTotal = lengthTotal + LenAux(k-1);
end
errorAv = errorTotal / lengthTotal;

[maxLenAux, whereMax] = max(LenAux);
thetaRef = thetaAux(whereMax);

for k=2:length(ISeq),
   errorInJump(k-1) = LenAux(k-1)*abs(sin(2*(mod(thetaAux(k-1)...
      - thetaRef, pi/2))));
   errorTotal  = errorTotal + errorInJump(k-1);
   if debugMode
      h=line(JSeq([k-1 k]),ISeq([k-1 k]));
      if whereMax==k-1
      set(h,'color',[1 0 0]);
         set(h,'lineWidth',2);
      end

      title(['e=' num2str(errorInJump(k-1)/LenAux(whereMax))...
           ', S=' num2str(abs(sin(2*(mod(thetaAux(k-1)...
           - thetaRef, pi/2)))))]);
      pause
   end
end
errorAvRef = errorTotal / lengthTotal;
errorMax = max(errorInJump/maxLenAux);

%==================================================================%
% End of file 'quadError.m'
```

```
function [r, CenterR, theta, d, base] = radsel(sizeA, sizeR, thetaIndex,
dispIndex)
%
% function [r, CenterR, theta, d, base] = radsel(sizeA, sizeR, thetaIndex,
dispIndex)
%
% 'sizeA' is the size of the original image
%
% 'sizeR' is the size of the Radon Transform matrix
%
% 'thetaIndex' is the angular information of the possibly detected PL
%
% 'dispIndex' (displacement index) is the distance to center
%  of the possibly detected PL
%
% 'r' is the linear band mask generated at (thetaIndex, dispIndex)
%
% 'CenterR' is the computed center of the image, as used by the
% Radon tranform routine.
%
% 'theta'  is the angle associated with the angular index 'thetaIndex'
%
% 'd'  is the distance in pixels associated with the displacement index
'thetaIndex'
%
% 'base' (=[P1 P2]) is the base point of the linear band mask

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'radsel.m'

%----------------------------------------------------------------%

NumOfDispl=sizeR(1); %2*ceil(norm(sizeA-floor((sizeA-1)/2)-1))+3;
NumOfAngles=sizeR(2);

% compute angle theta
thetaIncr=pi/NumOfAngles;
ZeroAngle=ceil((NumOfAngles+1)/2);
if thetaIndex>=ZeroAngle
   theta=(thetaIndex-ZeroAngle)*thetaIncr;
else
   theta=-thetaIncr*(ZeroAngle-thetaIndex);
end
cosTheta=cos(theta);
sinTheta=sin(theta);
```

```
% compute distance from origin
ZeroDispl=ceil((NumOfDispl+1)/2);
K=NumOfDispl/(2*ceil(norm(sizeA-floor((sizeA-1)/2)-1))+3);
d=(dispIndex-ZeroDispl)/K;

r=zeros(sizeA);

% compute origin
CenterXY = floor((sizeA+1)/2);
CenterR=[1+sizeA(1)-CenterXY(1) CenterXY(2)]  - [0.5 0.5];

% compute base point, closest point to the origin on the line
base=CenterR - d*[cosTheta sinTheta];

diagLength=sqrt(sizeA*sizeA');

for k=-(diagLength/2 + 1):0.2:(diagLength/2 + 1),
   pointNow=round(base + k*[-sinTheta cosTheta] + [0.5 0.5]);
   if (pointNow(1)<=sizeA(1))&(pointNow(2)<=sizeA(2))&(1<=min(pointNow))
      r(pointNow(1),pointNow(2))=1;
   end
end

r=filter2(ones(3,3),r,'same');
r(find(r>0))=1;

%=================================================================%
% End of file 'radsel.m'
```

```
function [P, Indexes] = rPartition(S)
%
% function [P, Indexes] = rPartition(S)
%
% Description: Creates partition 'P' from set of sets 'S',
%              such that:
%
% S{i} and S{j} will are included in the same
% partition P{k} if and only if intersect(S{i},S{j})
% is not empty.
%
% S{Indexes{k}}, with 1 <= k <= number of proper sets
% in partition 'P', are the sets of 'S' which merged
% into P{k}.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'rPartition.m'

%------------------------------------------------------------------%

n=length(S);

i=1;
EmptyList=[];

Indexes={};

for k=1:n,
   if isempty(S{k})
      EmptyList=[EmptyList k];
      Indexes{k}=[];
   else
      Indexes{k}=[k];
   end
end

while i < n,
   j = i + 1;
   while j <= n,
      if isempty(intersect(S{i},S{j}))
         j = j + 1;
      else
         S{j}=union(S{i},S{j});
         Indexes{j}=union(Indexes{i},Indexes{j});
         S{i}=[];
         Indexes{i}=[];
         EmptyList=[EmptyList i];
         j = n+1;
      end
   end
   i = i + 1;
end;

nonEmptyClusters = setdiff([1:n],unique(EmptyList));
P=S(nonEmptyClusters);
Indexes=Indexes(nonEmptyClusters);

%==================================================================%
% End of file 'rPartition.m'
```

```
function [Seg, Rest] = segm(Imag, N)
%
% Description: Get next segmented region from image 'Imag'.

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'segm.m'

%----------------------------------------------------------------%

Rest=Imag;
I=find(Imag > 0);

if length(I)>0
   [R,C]=ind2sub(size(Imag),I(1));
   [Seg,IDX] =  bwselect(Imag,C,R,N);
   Rest(IDX)=0;
else
   Seg=[];
end

%================================================================%
% End of file 'segm.m'
```

```
function [Building, figHandle] = ...
   selectbuildingcandidates(A, imageBackground, contourOnly,...
   BuildingCandidate, PLinBuild, cycleSummary, shapeError,...
   shapeMaxError, sFrac, numBuildingsInCluster, sizeA, debugMode)
%
% Description: Selects building candidate countours acording
%              by thresholding error measurers with non-increasing
%              functions heuristically adjusted.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'selectbuildingcandidates.m'

%------------------------------------------------------------------%

if imageBackground
   figHandle = imagesc(double(A)/255, [0 1]); axis image;...
      colormap(gray); plotColor=[0 1 0];
else
   figHandle = image(uint8(255*ones(sizeA))); axis image;...
      colormap(gray); plotColor=[0 0 0];
end

totalNumOfBuildings = 0;

for k=1:length(numBuildingsInCluster),

    for m=1:numBuildingsInCluster(k),

       if ~isempty(BuildingCandidate{k,m})
          ISeq = BuildingCandidate{k,m}(:,1);
          JSeq = BuildingCandidate{k,m}(:,2);

          if debugMode
             h=line(JSeq,ISeq);
              set(h,'LineWidth',2)
             set(h,'Color',[0 1 0])
             title(['Cluster=' int2str(k) '  #PL='...
                   int2str(length(PLinBuild{k,m})) '  Build=' int2str(m)...
                     '  shErr=' num2str(shapeError{k,m})...
                     '  shMax=' num2str(shapeMaxError{k,m})...
                     '  sFrac=' num2str(sFrac{k,m})])
             % pause
          end

          LB = length(PLinBuild{k,m});
          % monotonic threshold function application
          if  (((LB <= 3)&(sFrac{k,m}>=0.75)&...
                 (shapeError{k,m} <= 0.5)&...
                 (shapeMaxError{k,m} <= 0.9))|...
                ((LB == 4)&(sFrac{k,m}>=0.75)&...
                 (shapeError{k,m} <= 0.5)&...
                 (shapeMaxError{k,m} <= 0.70))|...
                ((LB >= 5)&(LB <= 9)&...
                 (sFrac{k,m}>=0.85)&...
                 (shapeError{k,m} <= 0.30)&...
                 (shapeMaxError{k,m} <= 0.35))|...
                ((LB >= 10)&(sFrac{k,m}>=0.85)&...
                 (shapeError{k,m} <= 0.30)&...
                 (shapeMaxError{k,m} <= 0.2)))

             if contourOnly & imageBackground
                h=line(JSeq,ISeq);
                set(h,'LineWidth',2)
                set(h,'Color', plotColor)
             else
                h=patch(JSeq,ISeq,plotColor);
             end

             totalNumOfBuildings = totalNumOfBuildings + 1;
             Building.PL{totalNumOfBuildings} = PLinBuild{k,m};
             Building.Cycle{totalNumOfBuildings} = cycleSummary{k,m};
             Building.OwnerCluster(totalNumOfBuildings)=k;
             Building.Contour{totalNumOfBuildings}.ISeq = ISeq';
             Building.Contour{totalNumOfBuildings}.JSeq = JSeq';
          else
             if debugMode
                h=line(JSeq,ISeq);
                set(h,'LineWidth',1)
                set(h,'Color', [0 0 1])
             end
          end
       end
    end
end

title([int2str(totalNumOfBuildings) ' building candidates found.'])

%===========================================================%
% End of file 'selectbuildingcandidates.m'
```

```
function [PLinLoop, Indexes] = ...
   seploops(A, PLinLoopOrig, PL, clusterFirst, debugMode)
%
%
% PLinLoop{k} = PLinLoopOrig{j}, for some j.
%
% Description:
%
% 'seploops' extracts cycles from the set PLinLoopOrig that don't
% contain other cycles in the same set PLinLoopOrig, thus eliminating
% some spuriuous cycle detections. Indexes{k} is the index in
% PLinLoopOrig{} of the k-th non-spurious cycle found.
%
% (arguments 'A' and 'PL' are only used for visualization plots)

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'seploops.m'

%------------------------------------------------------------------%

if clusterFirst
   % then merge those which intersect
   [PLinLoopMerged, MergedIndexes] = rPartition(PLinLoopOrig);
   PLinLoop = {};
   Indexes = [];
   T = 0;
   for k=1:length(PLinLoopMerged),
      ClusteredCycles = PLinLoopOrig(MergedIndexes{k});
      [P, Ind] = fPartition(ClusteredCycles);
      for j=1:length(Ind),
         T = T+1;
         PLinLoop{T}=P{j}; %=LoopCluster{Ind(T)};
         Indexes{T}=MergedIndexes{k}(Ind(j));
      end
   end
else
   PLinLoop = PLinLoopOrig;
   Indexes = [];
end

if debugMode
   T=0;
   for k=1:length(PLinLoop),
   if ~isempty(PLinLoop{k})
       T=T+1;
          plotwithlines(uint8(255*ones(size(A))),...
            {PL(:,PLinLoop{k})},[2 ],{[0 1 0]})
      if T > 1
         axis(v)
         end
      grid on
          if clusterFirst
            title([int2str(T) ': ' int2str(length(Indexes{T}))...
                  ' overlapping loops'])
         else
            title([int2str(T) ': [' int2str(PLinLoop{k})...
                  '] formant PL'])
         end

      pause
         v = axis;
      end
   end
end

%==================================================================%
% End of file 'seploops.m'
```

```
function [h, Proj]=sigdistoline(P,Line)
%
% Description:
%
% Computes the distance from point 'P' to a given line.
%
% 'Line' is a primitive line in the format:
% [theta, d, base, LimitI, LimitJ]

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'sigdistoline.m'

%----------------------------------------------------------------%

theta=Line(1);
d=Line(2);
base=Line(3:4);
Center = base(:)' + d*[cos(theta) sin(theta)];

Disp=P(:)'-Center;
Y=-Disp(1);
X=Disp(2);
YSinXCos=Y*sin(theta) + X*cos(theta);
XProj=cos(theta)*YSinXCos - d*sin(theta);
YProj=sin(theta)*YSinXCos + d*cos(theta);
Proj=Center+[-YProj XProj];

h=sign(Proj(1) - P(1))*norm(Proj-P(:)');

%================================================================%
% End of file 'sigdistoline.m'
```

```
function [loop, PLinLoop, h] = smartFindCycles(G, A, PL, IJCoordinates,
debugFlag)
%
% Description: Find cycles in the graph G and computes
%              polygons associated with each of them.

%================================================================%
%                                                                %
% COMPUTER-AIDED RECOGNITION OF                                  %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                      %
%                                                                %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe  %
%                                                                %
% Department of Computer Science                                 %
% Naval Postgraduate School, September 1999                      %
%                                                                %
%================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'smartFindCycles.m'

%----------------------------------------------------------------%

sPL = [];

loop={};
PLinLoop = {};

n = size(PL,2);

searchSet = [1:2:2*n];

while ~isempty(searchSet),
   otherVerticeOfPL = find(G(searchSet(1),:)==1);
   InitialPath = [otherVerticeOfPL searchSet(1)];

   k = floor((searchSet(1)-1)/2)+1;

   [loop{k}, h] = loopfromPL(InitialPath, 7, IJCoordinates);
   PLinLoop{k} = unique(floor((loop{k}-1)/2)+1);

   if length(PLinLoop{k}) < 3
      PLinLoop{k} = [];
   end

   searchSet = searchSet(2:length(searchSet));

   if ~isempty(PLinLoop{k})
      searchSet = ...
         setdiff(searchSet, setdiff(loop{k}, otherVerticeOfPL));
      disp(['Searching for cycles: '...
            num2str(round(1000*length(searchSet)/n)/10) '% done.'])

      if debugFlag
         PLinLoop{k}
         sPL = [sPL PL(:,PLinLoop{k})];

         plotwithlines(A, {PL(:,PLinLoop{k}) PL(:,k)},...
            [2 2], {[0 1 0][1 0 0]})
         v1=axis;
         title(['loop: [' int2str(loop{k}) '] g='...
            int2str((InitialPath(1)>InitialPath(2))+1)])
         xlabel(['  h=' int2str(h)]);

         pause

         v=axis;
         if sum(v==v1)~=length(v)

            for m=3:length(loop{k}),
               plnow = floor((loop{k}(m)-1)/2)+1;
               plotwithlines(A, {PL(:,PLinLoop{k}) PL(:,plnow)},...
                  [2 2], {[0 1 0][1 0 0]})
               title(['loop: [' int2str(loop{k}) '] g='...
                     int2str((InitialPath(1)>InitialPath(2))+1)])
          othernodenow = find(G(loop{k}(m),:)==1);
             xlabel(['Node=' int2str(loop{k}(m))...
                     '  Jumped with G='...
                     int2str(double(G(loop{k}(m-1),othernodenow)))]);
               axis(v)
               pause
            end
         end
      end
   end
end

if debugFlag
   plotwithlines(A, {sPL}, 2, {[0 1 0]})
end

%=================================================================%
% End of file 'smartFindCycles.m'
```

```
function [UsedPixels, LineDescription] = xlines(r, theta, disp)
%
%  [UsedPixels, LineDescription] = xlines(r, theta, disp)
%
%  r binary image
%
% Description: Computes the best line passing through the on-pixels
%              of each segmented region in r.

%==================================================================%
%                                                                  %
% COMPUTER-AIDED RECOGNITION OF                                    %
% MAN-MADE STRUCTURES IN AERIAL PHOTOGRAPHS                        %
%                                                                  %
% Luiz Alberto Cardoso, under supervision of Prof. Neil C. Rowe    %
%                                                                  %
% Department of Computer Science                                   %
% Naval Postgraduate School, September 1999                        %
%                                                                  %
%==================================================================%

% Programing Language: Matlab 5.3
% Operational System:  Windows NT 4.0
%
% This file named:    'xlines.m'

%------------------------------------------------------------------%

Rest=r;
Seg=zeros(size(r));
s=0;
LineDescription = [];
UsedPixels=[];

% compute origin
CenterXY = floor((size(r)+1)/2);
CenterR=[1+size(r,1)-CenterXY(1) CenterXY(2)]  - [0.5 0.5];

% compute base point, closest point to the origin on the line
base=CenterR - disp*[cos(theta) sin(theta)];
while ~(max(max(Rest))==0),
   [Seg, Rest] = segm(Rest,8);

   % test if Seg is plausible line segment
   if lineseg(Seg)
      % if it is: (1) increment s, B{s} <-- Seg
      %           (2) annotate parameters

      lineParms = bestline(Seg, CenterR); %, bigMask, auxSeg);
      angleOfThisSeg=lineParms(1);
      if (abs(cos(angleOfThisSeg-theta)) > cos(pi/5))
         LineDescription = [LineDescription lineParms];
         s=s+1;
         UsedPixels{s} = find(bwmorph(Seg,'spur',2)>0);
      end
   end
end

%==================================================================%
% End of file 'xlines.m'
```

LIST OF REFERENCES

1. Short, Sr., Nicholas M., *The Remote Sensing Handbook*, Goddard Space Flight Center, NASA, May, 1999. [http://rst.gsfc.nasa.gov/].

2. Smith, J., "Eyes over Kosovo", ABC News, 7 April, 1999. [http://more.abcnews.go.com/sections/tech/closerlook/recontech990407.html].

3. Boyle, A., and Windrem, R., "Spies on the watch for atrocities", MSNBC News, 26 March 1999. [http://www.msnbc.com/news/253629.asp].

4. Espuche, A. Garcia, Corboz, A, Bogdanovich B., and Solà-Morales I., *Ciudades: del globo al satélite*, Centre de Cultura Contemporània de Barcelona y Editorial Electa, Barcelona, 1994.

5. Moore, W., "About Ace Aerial", Ace Aerial Photography, Inc., 14 May 1999. [http://www.aceaerialphoto.com/AboutAce.html].

6. Federation of American Scientists, "Introduction to Imagery Intelligence", Intelligence Resource Program, 1997. [http://www.fas.org/irp/imint/imint_101.htm].

7. Mazour A. and King, S., "Design and Development of Human Equivalent Inspection System", Canpolar East Inc., September 1999. [http://www.vetech.com/photonics.htm].

8. Earth Resource Surveys Inc., "IKONOS Image Products", October 1999. [http://www.ersi.bc.ca/ikonos.html].

9. Microsoft Research, 1998. [http://terraserver.microsoft.com/].

10. Aerial Images, Inc., "Spin-2 Satellite Imagery",1999. [http://www.spin-2.com/].

11. Radon, J., English translation of "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten", Ber. Verh. Sächs. Akad. Wiss. Leipzig, Math-Nat., Vol. 69, pp. 262-277, 1917, in Deans S., *The Radon Transform and Some of Its Applications*, Appendix A, John Wiley & Sons, 1983.

12. Deans S., *The Radon Transform and Some of Its Applications*, John Wiley & Sons, 1983.

13. Ablameyko S., Lagunovsky D., "Aerial Images: from Straight Lines to Rectangles", *Proceedings of SPIE Conference on Visual Communication and Image Processing*, Chicago, Vol. 2308, pp. 2040-2048, 1994.

14. Lagunovsky D., Ablameyko S., "Rectangle-Shaped Object Detection in Aerial Images", *Proceedings of SPIE Conference on Visual Communication and Image Processing*, Taipei, Vol. 2501, pp. 1566-1574, 1995.

15. Canny, J., "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, 679-698, 1986.

16. Heath, M., Sarkar, S., Sanocki, T. and Bowyer, K., "Comparison of Edge Detectors. A Methodology and Initial Study", *Computer Vision and Image Understanding*, vol. 69, no. 1, January 1998.

17. Somka, M., Hlavac, V. and Boyle, R., *Image Processing Analysis and Machine Vision*, Chapman & Hall, London, 1993.

18. Cormen, H. T., Leiserson, C.E., and Rivest R.L., *Introduction to Algorithms*, Chapter VI, The MIT Press, 1990.

19. Hagan, M. T., Demuth, H. B., Beale, M., *Neural Network Design*, Chapter 11, PWS Publishing Company, 1995.

20. Cardoso, L. A., Rowe, N. C., "Report in Computer Vision - A Directed Study: Computer –Aided Interpretation of Aerial Photographs", Internal Report, Department of Computer Science, Naval Postgraduate School, Monterey, March 1999.

21. Lin, C. and Nevatia, R., "Buildings Detection and Description from Monocular Aerial Images", *ARPA96*, pp. 461-468, 1996.

22. Lin, C. and Nevatia, R., "Buildings Detection and Description from a Single Intensity Images", *Computer Vision and Image Understanding*, Vol. 72, No. 2, pp. 101-121, November, 1998.

23. Paparoditis, N., "Building Detection and Reconstruction from Mid- and High-Resolution Aerial Imagery", *Computer Vision and Image Understanding*, Vol. 72, No. 2, pp. 122-142, November, 1998.

24. Kuttikkad, S. "2-D Site Models from Single-pass SAR Image", 17 April, 1997. [http://www.cfar.umd.edu/~shyam/urban.html].

BIBLIOGRAPHY

Gupta, Madan M. and Knopf, George K., editors, *Neuro-Vision Systems Principle and Applications* (a collection of 47 selected papers in neuroscience and vision), IEEE Press, 1994.

Lin, C., Huertas, A. and Nevatia, R., "Detection of Buildings from Monocular Images", *Ascona95*, pp. 125-134, 1995.

Pratt, William K., *Digital Image Processing*, Second Edition, John Wiley & Sons, Inc., 1991.

Ritter, Gerhard X. and Wilson, Joseph N. *Handbook of Computer Vision in Image Algebra*, CRC Prress, , 1996.

Schutte, K., "Recognition of Buildings from Aerial Images", in: J.A.C. Bernsen, J.J. Gerbrands, A.A. Hoeve, A.W.M. Smeulders, M.A. Viergever, A.M. Vossepoel (eds.), *Third Quinquennial Review 1991-1996 Dutch Society for Pattern Recognition and Image Processing*, NVPHBV, Delft, pp. 211-225, 1996.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. National Reconnaissance Office 1
   14675 Lee Road
   Chantilly, Virginia 20151-1715

4. Professor Dan Boger 1
   Chairman, Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Professor Neil C. Rowe, Code CS/Nr 3
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Prof. Roberto Cristi, Code EC/Cx 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

7. Prof. Robert B. McGhee, Code CS/Mz 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

8. Prof. Craig W. Rasmussen, Code MA/Ra 1
   Department of Mathematics
   Naval Postgraduate School
   Monterey, California 93943-5121

9. Prof. Carlos F. Borges, Code: MA/Bc 1
   Department of Mathematics
   Naval Postgraduate School
   Monterey, California 93943-5121

10. Director, Instituto de Pesquisas da Marinha..
Rua Ipiru 2, Ilha do Governador
21931-090 Rio de Janeiro - RJ
BRAZIL

11. Head of Research, Instituto de Pesquisas da Marinha.. 1
Rua Ipiru 2, Ilha do Governador
21931-090 Rio de Janeiro - RJ
BRAZIL

12. Diretoria de Ensino da Marinha 1
via: Brazilian Naval Commission
5130 MacArthur Boulevard, NW
Washington, D.C. 20016-3344

13. Director, Diretoria de Engenharia Naval.. 1
Rua Primeiro de Março, 118 - 10° andar - Centro
20.010-000 Rio de Janeiro - RJ
BRAZIL

14. Diretoria de Sistemas de Armas da Marinha, Library.. 1
Rua Primeiro de Março, 118 - 19° andar - Centro
20.010-000 Rio de Janeiro - RJ
BRAZIL

15. Instituto Militar de Engenharia, Library... 1
Praça General Tibúrcio 80, Praia Vermelha
22290-270 Rio de Janeiro - RJ
BRAZIL

16. Centro Técnico Aeroespacial, Library 1
Praça Mal. Eduardo Gomes 50, Vila das Acácias
12228-904 São José dos Campos - SP
BRAZIL

17. Instituto Nacional de Pesquisas Espaciais... 1
Coordenação-Geral de Observação da Terra
Av. dos Astronautas, 1758 – Jardim da Granja
12201-970 São José dos Campos – SP
BRAZIL

18. CC(EN) Luiz Alberto Lisboa da Silva Cardoso.. 5
Rua Ipiru 2, Ilha do Governador
21931-090 Rio de Janeiro - RJ
BRAZIL